Final

PKI Specifications to Support the DoE Travel Manager Program

National Institute of Standards and Technology
Computer Security Division

August 15, 1996

William E. Burr
Donna F. Dodson
Noel A. Nazario
W. Timothy Polk

Final

PKI Specifications to Support the
DoE Travel Manager Program
August 12, 1996

1. INTRODUCTION 1

1.1 System Overview 1
1.2 Concept of Operations 1

2. INFRASTRUCTURE COMPONENT SPECIFICATIONS 2

2.1 Certification Authority (CA) 2
2.2 Organizational Registration Authority (ORA) 9
2.3 Directory Service (DS) 12

3. CERTIFICATE PROCESSING MODULE 13

3.1 Functional Specification 14
3.2 Technical Specifications 15

4. CRYPTOGRAPHIC MODULE REQUIREMENTS 17

4.1 CA Cryptographic Module Requirements 17
4.2 ORA Agent Cryptographic Module Requirements 17
4.3 User Hardware Cryptographic Module Requirements 17
4.4 User Software Cryptographic Module Requirements 18

5. SUPPORT APPLICATIONS 18

5.1 Initial Certificate Request 18
5.2 Certificate Renewal 18
5.3 Certificate Revocation 19

6. DATA FORMATS 19

6.1 Certificate Format 19
6.2 Certificate Revocation List (CRL) 29
6.3 Certification Path Validation 36
6.4 Transaction Message Formats 37
6.5 PKI Transactions 43

7. REFERENCES 50

APPENDIX A - X.509 V3 CERTIFICATE ASN.1 52

APPENDIX B - CERTIFICATE AND CRL EXTENSIONS ASN.1 56

APPENDIX C - API FOR CERTIFICATE PROCESSOR 64

APPENDIX D - CRYPTOGRAPHIC APPLICATION PROGRAMMING INTERFACE 71

APPENDIX E - SAMPLE PERFORMANCE SPECIFICATIONS 72

APPENDIX F - CA KEY PAIR UPDATE 76

Definitions, Terms, and Acronyms

Abstract Syntax Notation 1 (ASN.1): an abstract notation for structuring complex data objects; in a PKI, ASN.1 is always used with the distinguished encoding rules (DER).

accredited: recognize an entity or person to perform a specific action; CAs accredit ORAs and user sponsors

agents: personnel authorized to act on the behalf of an infrastructure component (CA, or ORA)

certify: the act of issuing a certificate

certificate: a structure defined by the X.509 authentication standard

certificate revocation list (CRL): a list of revoked but unexpired certificates issued by a CA

certification authority (CA): CAs issue certificates to users and other CAs. CAs issue CRLs periodically, and post certificates and CRLs to a directory service

Distinguished Encoding rules (DER): rules for encoding ASN.1 objects

digital signature: a data unit that allows a recipient of a message to verify the identity of the signatory and integrity of the message.

Digital Signature Algorithm (DSA): the digital signature algorithm specified in FIPS PUB 186.

directory service (DS): a distributed database service capable of storing information, such as certificates and CRLs, in various nodes or servers distributed across a network.

hash: a non-reversible method for reducing any amount of data to a fixed size and unique value.

message digest: the fixed size result of hashing a message.

operators: personnel who perform backups, review audit logs, and generally maintain the system.

organizational registration authority (ORA): an entity that acts an intermediary between the CA and user; the CA trusts the ORA to verify the binding between a user identity and their public key.

user registration: notification to a CA that a user should be issued a certificate.

user sponsor: a person who is permitted to register users with a particular CA.

Draft DoE PKI Specifications

1. Introduction

1.1 System Overview

This specification defines a public key infrastructure (PKI) and associated client services to support the DoE Travel Manager program. The infrastructure consists of a single certification authority (CA), multiple organizational registration authorities (ORAs), and a distributed directory service. This infrastructure will issue X.509 V3 certificates and V2 CRLs. The DoE PKI will be a complete PKI on its own, but is designed to join the emerging Federal PKI as it is deployed.
This is shown as Figure 1.

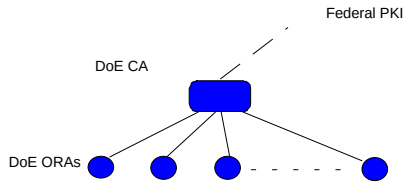

Figure 1**. DoE PKI and emerging Federal PKI**

Users will execute travel applications on their local system, using hardware or software cryptographic modules as required by their roles. In addition to the cryptographic module (which supports key generation, signature generation, and verification), a certificate processing module and several PKI support applications are required. The certificate processing module will obtain certificates and CRLs from the directory and verify certification paths. The PKI support applications will permit users to apply for their initial certificates, electronically renew certificates, and revoke certificates.

1.2 Concept of Operations

This section provides a brief overview of the concept of operations for DoE's public key infrastructure, and is intended to ease comprehension of the following specification.

DoE employees who will participate in the DoE Travel System will be registered with the DoE CA. The CA will generate distinguished names for each registered user and determine which type of cryptographic module is appropriate for the user. The CA will notify the employees to register for Travel training; software users will receive a software cryptographic module and instructional materials.

Software users will generate key materials on their own systems and bring the public key to the training session on a diskette. Hardware users generate their keys on hardware tokens. Both classes of users will present proper identification to the ORA after the training session; the ORA will request an initial certificate on behalf of the user.[1] The ORA

[1] For users who are not involved in travel, the training session prior to presentation of credentials and key materials to the ORA may be omitted.

will provide the user's certificate, the DoE CA's certificate, and the addresses of the CA and directory server to the user.

After initial certification, users will be able to generate and verify digital signatures. Verification of certification paths will be performed by finding a trust chain that begins with the DoE CA.

2. Infrastructure Component Specifications

This section specifies the components of a public key infrastructure in support of the DoE Travel System. Functional, security, and technical specifications are provided for each infrastructure component (i.e., the CA, ORAs, and the DS). The following specifications define transactions involving infrastructure components and the archiving requirements imposed upon the components.

2.1 Certification Authority (CA)

The Travel System CA shall generate, revoke, publish, and archive certificates. The CA shall rely upon a Directory Service (DS) to make certificates and CRLs available to all users. The CA shall archive all transactions (e.g., certificate requests, and revocation requests.) The CA shall incorporate facilities to conduct system backups, and maintain a separate audit log for the monitoring and tracking of security incidents. The operation and design of the CA shall be fully documented to allow operators thorough control and understanding of the implementation and the specification of any future enhancements.

2.1.1 CA Functional Specifications

The CA shall perform the following functions:

- Authenticate CA agents and systems operators;
- Enforce dual control;
- Generate and verify signatures using the DSA;
- Generate its own public-private key pair;
- Execute tests for the quality of the public key parameters (i.e., *p, q,* and *g*) identified in FIPS PUB 186;
- Request a CA certificate and establish cross-certificates;
- Create, deliver, and post subordinate certificates;
- Generate or obtain unique distinguished names for new users;
- Ensure that the subject of the certificate possesses the corresponding private key for every certificate issued without gaining access to the actual private key;
- Accept certificate revocation requests from users and other ORAs;
- Validate revocation requests and revoke certificates;
- Create, maintain, and post CRLs;
- Maintain required local information, such as the contents of CA name space;
- Maintain a list of accredited ORAs and user sponsors;
- Maintain record of all certificates issued and the number of renewals permitted for each certificate;

- Maintain record of all CRLs issued;
- Create and maintain system audit logs; and
- Generate or obtain time stamps.

Initialization

The CA shall be able to generate and test values for the DSA parameters *p*, *q*, and *g* as specified in FIPS 186. These parameters shall be used by the CA and all its subordinates in key generation. The size of the prime modulus *p* shall be 1024 bits for all CAs, ORAs, and users. The values of parameters *p*, *q*, and *g* shall be configurable. At a minimum, the quality tests for DSA parameters shall include:

- primality of p;
- primality of q;
- q divides evenly in p-1;
- $g > 1$;
- $y < p$.

The CA will generate its own key pair and publish its own certificate. It is expected that the DoE Travel System CA will be incorporated to the Federal PKI in the future, therefore it shall be able to perform cross certification. The CA's cryptographic module will cryptographically split the private key for backup purposes and the partial keys will be placed under the control of separate CA agents for backup purposes.

At initialization time, the CA shall archive the following items:

- DSS parameters;
- self issued certificate or certificate issued by parent CA;
- CA and parent CA Ids;
- assigned name space; and
- any other restrictions imposed on the CA.

If the CA is joining an existing hierarchically managed infrastructure, the CA will also generate a CA Certificate Request and transmit it to its superior CA.[2] This transaction will be archived, including the response from the superior CA.

Accrediting ORAs and User Sponsors

The CA shall maintain a database of accredited ORAs. Accredited ORAs can vouch for the identity of users requesting certificates. Some ORAs will, in addition, be permitted to vouch for the identity of ORA agents for other ORAs.

The CA shall maintain a database of accredited user sponsors. User sponsors register users by providing the CA with the names and organizational information of individuals that need to obtain certificates. Sponsors shall themselves obtain certificates that will be used to sign user name lists provided to the CA.

[2] The Federal PKI Technical Working Group has defined a hierarchically managed structure for the Federal PKI.

The CA shall archive all additions and deletions from the ORA and user sponsor databases.

User Registration

Upon receipt of a user name list, the CA shall verify that the signer is an accredited user sponsor. Using the information submitted by the user sponsors, the CA shall generate or obtain distinguished names and put together certification packages. These packages will be delivered to users along with the software for the generation of keys and initial certification requests. The distinguished names, certification package serial number, and other user information shall be retained by the CA to verify future certificate requests.

The distinguished names shall be created using X.500 name subordination. Distinguished names contain alphanumeric strings that uniquely identify certificate holders. The CA shall establish an appropriate name space, following guidelines from the U.S. Government's Electronic Messaging Program Management Office - Electronic Directory, and only issue certificates for subject names within it.

The CA shall archive all submitted user name lists, even if they are rejected. The CA shall archive all distinguished names with the corresponding users names and organizational information.

Issuing Certificates

The CA shall support two forms of certification requests: *initial* and *renewal*. In an initial request, the identity of the requestor is established in person to an ORA. After examining identification evidence, the ORA vouches for the user's identity and the binding to the public key. In a renewal request, the established identity of the requestor is perpetuated with the request.

The CA shall accept initial certificate requests from ORAs vouching for users and other ORAs. CA shall also accept certificate renewals from both users and ORAs. Certificate requests shall be signed by the ORA and include the user's distinguished name, public key, organization, time stamp, and contact information (electronic mail address, phone number, and postal address).

Upon receipt of a certificate request from an ORA, the CA shall:

1. verify that the request comes from an accredited ORA;
2. verify the ORA's signature;
3. verify that the user/ORA was registered by an accredited user sponsor;
4. verify that the information on the request matches that provided by the sponsor;
5. form and sign a new user/ORA certificate;
6. post the new certificate on the DS; and
7. send a Certificate Request Response to the ORA containing the new certificate and the CA certificate.

The ORA will deliver the new certificate and the CA's certificate to the user. If the CA rejects the certificate request, it shall report the failure to the ORA stating the reason.

Upon receipt of a certificate renewal request from an entity, the CA shall:

1. verify that the renewal request was signed by the private key corresponding to the entity's unexpired unrevoked certificate;
2. verify that a second nested signature was made with the private key corresponding to the new public key;
3. verify that the certificate is eligible for renewal (by examining flags in the certificate or querying a local database);
4. form and sign a new user certificate;
5. post the new certificate on the DS; and
6. send a Certificate Request Response to the entity containing the new certificate.

If the CA rejects the certificate renewal request, it shall report the failure to the entity stating the reason. The CA shall log all certificate and renewal requests and the CA responses, even if the transactions are rejected.

The CA shall issue X.509 version 3 certificates. The fields and extensions utilized, and the values assigned to them, shall be in accordance with Section 6.1. After generating and signing the certificate, the CA shall send the certificate to the directory service and the requestor. The CA shall include a directory user agent (DUA) to access the directory service. The CA's DUA shall support authentication of the CA to the directory service.

Cross Certification

The CA shall issue certificates to other CAs with appropriate constraints. One CA initiates the cross certification process through procedural mechanisms as defined in the CA's Operational Policy. After agreeing to cross certify, the CAs exchange public keys and the associated parameters through out-of-band channels.

The initiating CA issues a certificate with appropriate constraints to the responding CA. The responding CA issues a certificate with appropriate constraints to the initiating CA. The CAs post the certificates to the directory service.

The CAs may optionally construct X.509 cross certificates from the new certificates and post them to the directory service as well.

The Travel System CA shall include mechanisms to support both the initiating and responding CA roles. The constraints placed upon CA certificates issued as a result of cross certification will be determined by policy. The cross certification mechanisms shall support specification of all constraints identified in section 6.1.

Revoking Certificates

The CA shall be capable of generating and issuing CRLs. The CA shall support issuing a single CRL for its name space and distribution point CRLs. The CA may optionally support incremental, or delta, CRLs. The types of CRLs issued will be determined by policy.

CRLs shall be generated from the previous CRL and approved pending certificate revocation requests. When a new CRL is generated, revoked unexpired certificates from

the previous CRL shall be carried over to the new CRL, and any certificates with approved pending certificate revocation requests shall be added to the new CRL. A certificate with an approved pending certificate revocation request shall be included in the next CRL even if it expires before the CRL is issued.

The CA may only revoke certificates it issued. The signer of the revocation request must either be an accredited ORA acting on behalf of the certificate holder or the holder's sponsor, or the holder of the certificate to be revoked. The CA shall validate the revocation request prior to including the certificate in a CRL.[3] If the revocation cannot be validated, or the source of the request was not one of the authorized parties, the revocation request shall be rejected.

The CA shall issue X.509 version 2 CRLs.[4] The fields and extensions utilized, and the values assigned to them, shall be in accordance with Section 6.2. After generating and signing the CRL, the CA shall send the CRL to the directory service.

CRLs and certificate revocation requests shall be reported and logged, even if they are rejected. Notifications of rejection shall also be logged.

Records keeping

The CA shall log the following certification activities: request to create a certificate, certificates issued, certificate requests rejected, request to revoke a certificate, revocation of a certificate, rejection of a revocation, generation of a CRL, and posting of a CRL to the DS. This information shall be stored off-line for archival purposes at least weekly. The actual frequency shall be established by the CA Operational Policy based on workload and perceived security threats. All archived information shall be maintained in a form that prevents unauthorized modification.

The CA shall include facilities to perform periodic, automated full and incremental backups to off-line storage.

The CA shall keep a separate audit log for the monitoring and tracking of security incidents.

Failure or Compromise Recovery

Steps shall be taken to minimize the possibility of compromise and corruption of the CA, its own databases, and the directory. Upon occurrence of a system compromise or failure that may affect the integrity of the system, the CA shall generate, or obtain if merged into the Federal PKI, a new certificate, issue the appropriate CRLs, and notify the affected parties of the need to re-authenticate to replace the compromised certificates.

To support compromise and failure recovery, the CA shall include facilities to re-issue previously issued certificates under a new CA key whenever possible. If the integrity of

[3] Validation of a revocation request shall include verification of the signature on the request. Out-of-band verification of revocation requests signed by ORAs may optionally be required by policy. The CA's revocation mechanism must be configurable to support such policy requirements.

[4] Version 2 CRLs correspond to the Version 3 certificate; the Version 2 certificate definition did not result in cretion of a new CRL format.

existing certificates can be established (e.g., through use of off-line archives), the CA may use these facilities to re-issue the user certificates with the new CA key.[5]

A key backup process, where separate CA agents hold separate key components, shall be provided to facilitate recovery from component failures which do not compromise the key or result in database corruption. This process will permit a CA to resume issuing certificates with the current private key using a new cryptographic module.

2.1.2 Security Specifications

The CA shall operate under the principles of split knowledge and dual control. The CA shall include a hardware cryptographic module which meets the requirements for CA cryptographic modules presented in Section 5.

Private keys shall be stored within the cryptographic module or encrypted using a FIPS approved confidentiality algorithm before being output. CA keys used for signing certificates shall never be exported in clear form and should reside in a cryptographic module operated under the principles of split control of two or more CA agents. At least two CA agents shall be required to activate the CA's cryptographic module for signing certificates; provisions shall be made for backup agents for cryptographic module activation. CA signature keys shall always be backed up under split control, i.e., no single agent shall be able to load the complete private key on any system without the cooperation of the other agents holding portions of the key. CA signature public keys shall be 1024 bits long; the validity period shall be configurable.

2.1.3 Transaction Set

Table 2-1 summarizes electronic transactions used in providing certificate management services to users of the Travel System. These transactions enable request, delivery, and revocation of ORA and user certificates, posting of certificates and CRLs on the DS, and the retrieval of certificates and CRLs from the DS for signature verification.
The CA shall process initial certification requests through its ORAs in the form of **CertReq** messages. The **CertReq** message is signed by the ORA in the **PKIProtection** structure. By signing the request, the ORA vouches for the identity of the user and confirms that the user is in possession of the corresponding private key. The CA responds to the ORA or user with a **CertRep** message. If the request was accepted, this message contains the new certificate. If the request was rejected, this message contains the error code.

The CA shall process certificate renewal requests in the form of **KeyUpdReq** messages. These messages are sent to the CA by the entity requesting the certificate. The message shall include the user's distinguished name, the serial number of their current certificate, the new public key, a proposed validity period, and a proposed key id. The message shall be signed twice according to the format in Section 6.4.1. The certificate renewal request shall be signed with the new private key; the **PKIHeader** and **PKIBody** shall constitute the

[5] Travel System users must obtain the CA's public key and parameters through out-of-band channels (e.g., from the ORA) to validate certification paths. The change will be tranparent to users who trust other CAs that have cross-certified with the Travel System CA.

message. The signed certificate request will be re-signed with the old private key corresponding to the entity's unexpired, unrevoked certificate from the CA. The second signature shall be calculated using the signed **PKIMessage** as the message. The CA shall respond to the requestor in the form of an **KeyUpdRep** message. This message will contain either a new certificate or a failure code. If issued, the certificate will include the user's distinguished name and the new public key. The CA is free to modify the proposed validity period and key identifier.

Table 2-1 - Travel System CA PKI Transaction Set

Transaction	Description	From	To
Initial Certificate Request	ORA submits a certificate request on behalf of an authenticated user	ORA	CA
	CA returns signed certificate or error message	CA	User, ORA
Certificate Revocation	ORA or certificate holder requests revocation of a certificate	User, ORA	Issuer CA
	CA responds with acceptance or rejection of the revocation request	Issuer CA	ORA, User
Certificate Renewal Request	doubly signed certificate request - new public key and current certificate # signed with new and old private keys	User, ORA	CA
	CA returns signed certificate or error message	CA	User, ORA
Post Certificate	CA posts a new certificate to the DS after strong authentication	CA	DS
Post CRL	CA posts a new certificate to DS after strong authentication	CA	DS
Retrieve Certificate	Query DS for an entity's certificate(s)	CA	DS
Retrieve CRL	Query DS for the latest CRL issued by a particular CA	CA	DS

The CA shall receive **RevReq** messages from ORAs or end entities. The **RevReq** message shall include the certificate serial number or the user's distinguished name and the key identifier. The CA shall respond with a **RevRep** message. This message shall include status and failure information, and may include additional details about the revoked certificate.

The Travel System CA shall maintain a log of all instances of the following transactions: certificate requests, certificate renewals, revocation requests, CRL generation, and posting of certificates and CRLs. All entries of the log shall be timestamped.

2.2 Organizational Registration Authority (ORA)

The O RA vouches for the identity of users requesting certification. Users request initial certification by appearing in person before an ORA for their parent CA and submitting a certificate request signed with the private key for the public key being certified. The format for the certificate request appears in Section 6.4. The ORA shall verify the user's signature on the request to ensure that the user possesses a complete key pair. An ORA agent verifies all the personal and affiliation information on the request and the signature according to the issuance policy for the type of certificate being requested. After the signature and the user's identity are verified, the ORA agent signs and sends an electronic certificate request to the CA.

The ORA requests certificate revocation on behalf of users who can not access their private key and organizations that wish to revoke an employee's certificate. The format of the revocation request is presented in Section 6.4. The ORA shall verify the identity of the requesting party. After the signature and the user's identity are verified, the ORA agent signs and sends an electronic revocation request to the CA.

The ORA function may be collocated with the CA or performed at a separate facility.

The operation and design of the ORA shall be fully documented to allow operators thorough control and understanding of the implementation and the specification of any future enhancements.

2.2.1 ORA Functional Specifications

ORAs shall perform the following functions:

- Provide users with the CA's public key and parameters;
- Authenticate ORA agents and system operators;
- Communicate electronically with parent CA to exchange transaction messages described in Section 2.2.3;
- Verify signatures on user certificate requests to ensure user possesses complete public-private key pair without gaining access to the actual private key;
- Process certificate revocation requests;
- Create and maintain system audit logs;
- Archive all transactions;
- Periodic system backups.

ORAs shall keep record of the identity of ORA agents, the dates and times when they operated the ORA, and the transactions (e.g., certificate requests, certificate revocations) they processed. Upon receipt of a certificate request, the ORA agent authenticates the user according to the appropriate certificate issuance policy, verifies the signature on the request using the public key in the certificate, signs the request, and sends it to the parent CA. The CA need not return the signed certificate to the user via the ORA, it may do so

directly. If the CA sends the newly signed certificate to the ORA, the agent shall copy it onto the user's diskette or load it into the user's hardware token along with the CA's certificate. The ORA shall always provide the CA's public key and parameters to the user requesting a certificate.

ORAs shall accept and verify in-person certificate revocation requests from the users. The ORA agent shall sign the requests and send them to the parent CA.[6] ORAs shall have a backup capability, be able to archive all transactions performed, and keep record of personal credentials reviewed.

2.2.2 ORA Security Specifications

ORAs shall authenticate ORA agents and system operators before performing any transactions or systems operations. The ORA shall include facilities for full and incremental backup, archiving PKI transactions, and system logs.

All operations of the ORA, except system backups, shall require an activated ORA agent's cryptographic module. The ORA agent's cryptographic module shall provide integrity and non-repudiation for PKI transactions and archives. Additional security requirements of the ORA, as specified in the DoE CA's Operational Policy, will be achieved through employment of physical security measures.

Requirements for ORA agents' cryptographic modules are specified in Section 5. ORA agents' signature public keys shall be 1024 bits long, the key validity period shall be configurable. If the system design permits system operators other than ORA agents to perform system operations, their cryptographic module shall meet the requirements for software cryptographic modules presented in Section 6.4.

2.2.3 Transaction Set

Table 2-2 gives the subset of electronic transactions used by the ORA in providing certificate management services to users of the DoE Travel System. These transactions enable request, delivery, and revocation of ORA and user certificates, and the retrieval of certificates and CRLs from the DS for signature verification.

The ORA shall receive initial certification requests on diskettes in the form of **CertReq** messages. The **CertReq** message is signed by the user in the **PKIProtection** structure. After reviewing the user's credentials and confirming that the user is in possession of the corresponding private key, the ORA will extract the **CertReqContent**, and create a new **CertReq** message with the ORA agent's name and signature. The ORA will send this message to CA. The ORA shall copy the Travel System CA's certificate to the user's diskette, along with configuration information (such as the CA's network address.)

The ORA may receive a **CertRep** message from the CA. If the request was accepted, the ORA shall copy the certificate to the user's diskette. If the request was rejected, the ORA will review the error code and may submit a new request.

---

[6] ORAs issuing certificates to hardware cryptographic module holders must be able to read and load selected information from and to hardware cryptographic tokens. The ORA shall not have access to user signature private keys.

The ORA shall perform revocation requests upon the request of authenticated users or their organizations. The ORA shall generate **RevReq** messages, including the certificate serial number or the user's distinguished name and the key identifier. The message shall be signed by the ORA agent. The CA shall respond to the ORA with a **RevRep** message. This message shall include status and failure information, and may include additional details about the revoked certificate. If the certificate is revoked, the ORA shall provide this information to the requestor. If the request is rejected, the ORA will review the error code and re-formulate the request.

Table 2-2 - Travel System ORA PKI Transaction Set

Transaction	Description	From	To
Initial Certificate Request	ORA submits a certificate request on behalf of an authenticated user	ORA	CA
	CA returns signed certificate or error message	CA	User, ORA
Certificate Revocation	ORA requests revocation of a certificate	ORA	Issuer CA
	CA responds with acceptance or rejection of revocation request	Issuer CA	ORA
Certificate Renewal Request	doubly signed request new public key and current certificate # signed with new and old private keys	ORA	Issuer CA
	CA returns signed certificate and CA's certificate	Issuer CA	ORA
	confirmation signed with old key	ORA	Issuer CA
Retrieve Certificate	Query DS for an entity's certificate(s)	ORA	DS
Retrieve CRL	Query DS for latest CRL issued by a particular CA	ORA	DS

ORAs shall maintain a log of all instances of the following transactions: Certificate Request, Certificate Renewal Request, Certificate Request Response, Key Compromise Report, and Certificate Revocation Request. All entries of the log shall be time stamped.

2.3 Directory Service (DS)

A directory service shall be provided that allows any Travel System user to retrieve certificates and verify the electronic signatures by any other user under the Travel System CA. The DS shall also store the most recent certificate revocation list (CRL) for the Travel System CA. It shall also allow users certified by other CAs to retrieve Travel System user certificates. This section contains the specifications for the Directory Service.

2.3.1 DS Functional Specifications

The DS shall perform the following functions:

- Store certificates and CRLs issued by the CA;
- Accept and respond to user queries for certificates and CRLs;
- Authenticate entities attempting to update or modify certificates or CRLs; and
- Prevent unauthorized access to the system running the service.

2.3.2 DS Security Specification

The DS shall authenticate all entities attempting to update or modify certificates and CRLs. Only the CA may create or update such entries. Access to the system running the service shall be strictly controlled and limited to the appropriate personnel.

2.3.3 DS Implementation

The assumption of the X.509 standard, and many of its derivative standards, is that certificates are available in X.500 directories and certification path verifiers find needed certification paths by directory queries. This is the ultimate goal of the Federal Public Key Infrastructure (FPKI), however full X.500 directory service may not be available in the near term for the use of the Travel System and establishment of a proper X.500 directory for the Travel System may not be a cost effective alternative. The Travel System may be designed so that some clients sign documents, but do not verify signatures and therefore do not require a signature verification capability and the capability to find certification paths. This section states the requirements for finding and making certification paths available to the certification path verifiers, where they are needed.

Travel System clients that must verify signatures, shall have an automated process for finding certification paths to process the signatures on travel documents. This certificate processing specification does not establish where signatures (and therefore certification path verification) are required in the Travel System. This automated process for finding certification paths may consist of (but is not limited to) one or more of the following:

(a) provision of an X.500 directory service to hold certification path and/or other travel information, with a user directory agent to access that information. The X.500 distinguished names contained in certificates are used to locate needed certificates. From the point of view of compatibility with the emerging Federal PKI, this is the most desirable solution;

(b) use of some other commercial directory product that may not be strictly X.500 compatible to access needed certificates. Alternative Name extensions may be used to facilitate accessing certificates;

(c) use of a World Wide Web server to provide certification path information. Alternative Name extensions may be used to provide URLs pointing to certificate information;

(d) use of an on-line database to supply certification path information. Access to the database may be via a network, or the database may be replicated at points where signatures must be verified.

(e) the Travel System may be designed so that clients provide the needed certificates attached to the documents that they sign.

For extensibility to future applications and integration with the FPKI, the process for certificate path verification and the process for finding certification paths shall be separated, so that it is possible to upgrade or replace the process for finding certification paths, without replacing the certification path verification process.

If the provided mechanism for finding certification paths requires alternative names (for example URL or Internet address) then the Travel System CA shall be capable of generating the appropriate alternative name in certificates and CRLs.

3. Certificate Processing Module

Applications will obtain PKI-supported security services through a *certificate processing module*. The application interface, defined in Appendix C, will provide a simple interface to security services. This module will request infrastructure services, such as issuing or revoking certificates, and obtaining current certificates and CRLs . The module will manage a local cache of certificates and CRLs to enhance performance. The certificate processing module shall verify certificate chains, generate certificate requests, and request certificate revocation. The certificate processing module will rely upon the availability of a cryptographic module to obtain required cryptographic services, such as generation and verification of digital signatures and key generation.

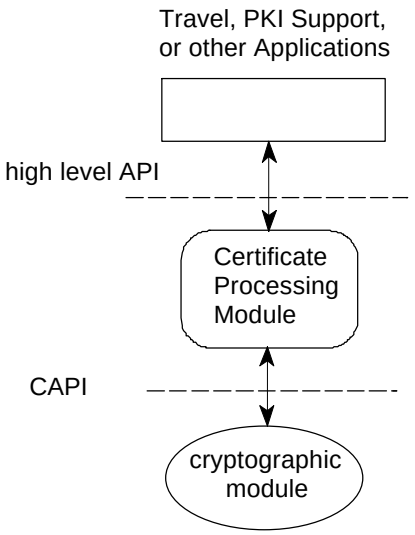

Figure 2**. Accessing PKI Services**

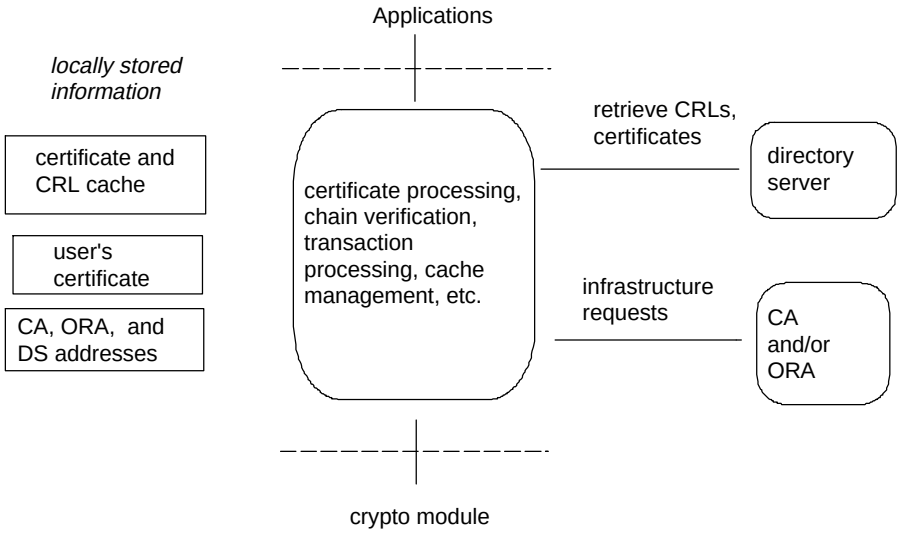

Figure 3**.** Certificate Processing Module

3.1 Functional Specification

The certificate processing module implements transactions with the PKI infrastructure components, manages a local information base, and processes certificate chains. The module also provides a high-level interface to basic cryptographic functions; these calls will be passed on to the cryptographic module (Section 5) for processing.

The certificate processing module shall perform transactions with the Travel System CA and the directory service. The certificate processing module shall manage a local information base including the user's certificate, common infrastructure information (e.g., common DSA parameters), system configuration information (e.g., addresses for directory servers, ORAs, and the CA), and a cache of certificates and CRLs to enhance performance. The certificate processing module shall protect the integrity of its information base with digital signatures.

The certificate processing module shall support the programming interface specified in Appendix C. The certificate processing module shall rely on the availability of a cryptographic module supporting the programming interface specified in Appendix D.

The certificate processing module shall be able to perform the following functions:

- Request certificate registration and renewal;
- Request certificate revocation;
- Query the DS for certificates and CRLs;
- Verify the path of a certificate to the DoE Travel System CA;
- Sign and timestamp information objects;
- Verify signed, timestamped information objects; and
- Manage a local certificate and CRL cache.

The certificate processing module shall be able to generate and transmit certificate registration, renewal, and revocation requests to its CA, and process the response. It shall use the DS for obtaining certificates and CRLs, and shall verify the signature of each entry retrieved. Certificates and CRLs obtained from the directory or from other sources are used for performing path verification. For efficiency, the certificate processing module shall cache CRLs and certificates where appropriate.

3.2 Technical Specifications

The certificate processing module shall:

- Generate directory service requests and interpret responses;
- Parse certificate and CRL formats as defined in Section 2;
- Verify certificate chains beginning with the Travel System CA;
- Generate and interpret PKI messages;
- Cache certificates and CRLs; and
- Manage local information, such as the addresses of the DS and CA.

The certificate processing module shall implement the transactions listed in Table 3-1. The certificate processing module shall generate messages where the From column specifies **User** or **Any**. The certificate processing module shall receive and interpret message where the To column specifies **Users**.

The certificate processing module shall implement the programming interface specified in Appendix C. The certificate processing module shall rely upon the services of a cryptographic module supporting the CAPI specified in Appendix D. The certificate processing module may rely upon the application to provide time and date information.

The certificate processing module shall support the generation of the following PKI message types:

- **CertReq**;
- **RevReq**; and
- **KeyUpdReq**.

The certificate processing module shall support the processing and interpretation of the following PKI message types:

- **CertRep**;
- **RevRep**; and
- **KeyUpdRep**.

The certificate processing module shall support the generation and processing of the PKI message types **PKIConfirm .**

The certificate processing module shall include a directory user agent (DUA) to retrieve certificates and CRLs from the directory service.

Table 3-1- User PKI Transaction Set

Transaction	Description	From	To
Initial Certificate Request	User submits certificate request on diskette to ORA along with credentials	User	ORA
	ORA submits a certificate request on behalf of an authenticated user	ORA	CA
	CA returns signed certificate or error message	CA	User, ORA
Certificate Revocation	User requests revocation of a certificate	User, ORA	Issuer CA
	CA responds with acceptance or rejection of the revocation request	Issuer CA	User, ORA
Certificate Renewal Request	doubly signed certificate request - new public key and current certificate serial number signed with new and old private keys	User	CA
	CA returns signed certificate and CA's certificate	CA	User
	signed with old key	User	CA
Retrieve Certificate	User queries DS for an entity's certificate(s)	User	DS
Retrieve CRL	User queries DS for the latest CRL issued by a particular CA.	User	DS

4. Cryptographic Module Requirements

For Travel System PKI components and clients the following functions shall occur within cryptographic modules:

- generation of random numbers;
- generation of DSS key pairs;
- protection of DSS private keys;
- generation of SHA-1 message digests;
- generation of DSS signatures; and
- verification of DSS signatures.

Cryptographic modules may be implemented using software, hardware or a combination of both. Cryptographic modules shall include validated implementations of DSS, SHA-1, and DES. The DoE Travel System shall allow the use of software cryptographic modules for regular users (e.g., travelers) and will require that hardware be used to produce the signatures by CAs, CA agents, ORA agents, and other users responsible for the operation and maintenance of systems running the travel application.

To allow the substitution of cryptographic modules in client systems, user cryptographic modules shall implement the Cryptographic Application Programming Interface (CAPI) specified in Appendix D.

4.1 CA Cryptographic Module Requirements

In addition to the general requirements above, CA cryptographic modules shall:

- be implemented in hardware and validated as complying with FIPS 140-1 level 3 requirements;
- require the action of two or more CA agents to activate so it can perform digital signature operations; and
- export the private key in cryptographically split fashion so that the independent key components can be placed under the control of separate CA agents for backup purposes.

4.2 ORA Agent Cryptographic Module Requirements

In addition to the general requirements above, ORA cryptographic modules shall:

be implemented in hardware and validated as complying with FIPS 140-1 level 1 and the level 3 identity-based operator authentication requirements;

- be bound to the identity of the ORA agent (as opposed to the ORA); and
- be unable to export the private signature key.

4.3 User Hardware Cryptographic Module Requirements

In addition to the general requirements above, user hardware cryptographic modules shall:

- authenticate the user;

- be validated as complying with FIPS 140-1 level 1 and the level 3 identity-based operator authentication requirements; and
- be unable to export the private signature key.

4.4 User Software Cryptographic Module Requirements

In addition to the general requirements above, user software cryptographic modules shall:

- authenticate the user;
- be validated as complying with FIPS 140-1 level 1 requirements;
- maintain the private key in encrypted form when not in use;
- always take care to overwrite system memory after using the signature key; and
- use DES in Cipher Block Chaining Mode to encrypt the private signature key and protect the DES key with a password at least eight characters long that includes one or more non-alphabetical characters.

5. Support Applications

DoE Travel users will require PKI specific functions to request, renew, or revoke certificates. Applications implementing these functions utilizing the high level API defined in Appendix C are required to support DoE travel users. These functions may be implemented as separate applications, or as a part of the DoE Travel applications.

5.1 Initial Certificate Request

This function will generate a DSS public-private key pair and create a PKI **CertReq** message requesting a v3 certificate with the signing algorithm **dsaWithSHA1** and the public key. The user will sign the message using the private key in the **PKIProtection** structure. The **CertReq** message will be written onto a diskette The user can take the diskette with their credentials to an ORA to obtain a certificate from the PKI.

The ORA shall store the CA's current public key and parameters on the diskette. The ORA may also store the user's certificate on the diskette, or the CA may return a **CertRep** message to the user. This message will contain the certificate or a reason code for the transaction failure. The protocol supporting this transaction is described in detail in Section 6.5, PKI Transactions.

5.2 Certificate Renewal

This function will generate a DSS public-private key and create a PKI **CertReq** message requesting a v3 certificate with the signing algorithm **dsaWithSHA1** and the public key. The user will sign the message twice; the first signature will be generated with the private key corresponding to the user's unexpired, unrevoked certificate. The second signature will be calculated on the message formed by the request and the first signature; the signature will be generated with the new private key corresponding to the public key in the certificate request.

The CA will return a **CertRep** message. This message will contain the certificate or a reason code for the transaction failure. This function must interpret the **CertRep** message

and update the Cryptographic Module information base as appropriate. The protocol supporting this transaction is described in detail in Section 6.5, PKI Transactions.

5.3 Certificate Revocation

Users may request revocation of their own certificates. To perform this function the user generates a **RevReq** message, signs it with the certificate to be revoked, and sends it to the CA. The CA responds with a **RevRep** message. This module must be able to interpret the message and update the Cryptographic Module information base as appropriate. The protocol supporting this transaction is described in detail in Section 6.5, PKI Transactions.

6. Data Formats

Basic data formats must be defined for interoperability of PKI components. The data formats include certificate, CRL, and transaction formats. These specifications include data formats for all transactions between infrastructure components, and between PKI clients and infrastructure components.

6.1 Certificate Format

The DoE Travel System shall use the X.509 V3 certificate format. Although the revision to ITU-T Recommendation X.509 that specifies the version 3 format is not yet published, the version 3 format has been widely adopted and is specified in American National Standards Institute X9.55-1995 [X9.55], and the Internet Engineering Task Force's Internet Public Key Infrastructure working document [PKIX1]. The X.509 version 3 certificate includes the following:

Version
Serial Number
Issuer Signature Algorithm
Issuer Distinguished Name
Validity Period
Subject Public Key Information
Issuer Unique Identifier (optional)
Subject Unique Identifier (optional)
Extensions (optional)
Issuer's Signature on all the above fields

6.1.1 Certificate Fields

The Abstract Syntax Notation One (ASN.1) definition of the X.509 certificate syntax is stated in Appendix A. For signature calculation, the certificate is encoded under the ASN.1 Distinguished Encoding Rules (DER). ASN.1 DER encoding is a tag, length, value encoding system for each element.

The following items specify the use of the X.509 v3 certificate. With the exception of the optional **subjectUniqueID** and the **issuerUniqueID** fields, the Travel System CA shall generate these fields and clients shall be capable of processing them in accordance with the

X.509 standard. The Travel System CA shall not issue certificates containing the optional **subjectUniqueID** and the **issuerUniqueID** fields.

Version

The **version** field describes the version of the encoded certificate. The value of this field shall be 2, signifying a version 3 certificate.

Serial number

The **serialNumber** is an integer assigned by the CA to each certificate. It shall be unique for each certificate issued by the Travel System CA (i.e., the issuer name and serial number identify a unique certificate).

Signature

The **signature** field contains the algorithm identifier for the algorithm used to sign the certificate. The **signature** field includes an **algorithmIdentifier**, which, in principle may be used to pass parameters. Certificates conforming to this specification shall be signed with the DSA algorithm, and the contents of the **algorithmIdentifier** field shall be as specified in Section 6.1.2. Certificates shall not include parameters in the **signature** field.

Issuer Name

The **issuer** field provides a globally unique identifier of the authority signing the certificate. The syntax of the issuer name is an X.500 distinguished name.

Validity

The **validity** field indicates the dates on which the certificate becomes valid (**notBefore**) and on which the certificate ceases to be valid (**notAfter**).

The **UTCTime** (Coordinated Universal Time) values included in this field shall be expressed in Greenwich Mean Time (Zulu) and shall express granularity to the minute. Seconds shall not be used. `UTCTime` shall be expressed as YYMMDDHHMMZ.

Subject Name

The purpose of the **subject** field is to provide a unique identifier of the subject of the certificate. The syntax of the subject name shall be an X.500 distinguished name.

Subject Public Key Information

The **subjectPublicKeyInfo** field is used to carry the public key and identify the algorithm with which the key is used. It includes the **subjectPublicKey** field and an **algorithmIdentifier** field with **algorithm** and **parameters** subfields. Certificates conforming to this interoperability specification shall use the DSA algorithm, and the contents of the **algorithmIdentifier** field shall be as specified in Section 6.1.2. The **parameters** subfield of

the **subjectPublicKeyInfo** field shall be the only method used to pass or obtain DSA parameters.

Unique Identifiers

The **subjectUniqueIdentifier** and **issuerUniqueIdentifier** fields are present in the certificate to handle the possibility of reuse of subject and/or issuer names over time. Certificates shall not include these unique identifiers.

Extension

The addition of the **extension** field is the principal change introduced to X.509 v3 certificates. Extensions have three components: **extnId**, that names the extension, **critical,** the criticality flag that specifies that the extension is critical or noncritical, and **extnValue,** the extension value. A certificate may contain any number of extensions, including locally defined extensions. If the criticality flag is set, a client shall either be able to process that extension, or shall not validate the certificate.

A set of standardized extensions has been developed in an amendment to the X.509 standard [DAM]. The use of these standardized extensions in conforming implementations is specified in Section 6.1.3 below.

SIGNED Macro

The actual signature on the certificate is defined by the **SIGNED** macro. See Appendix A for the ASN.1 definition of the **SIGNED** macro and associated operators. The signature includes an **algorithmIdentifier** that identifies the algorithm used to sign the certificate. Although this **algorithimIdentifier** field includes a **parameters** field that can, in principle, be used to pass the parameters used by the signature algorithm (see Section 6.1.2), it is not itself a signed object. The **parameters** field of the certificate signature shall not be used to pass parameters. The parameters used to validate a signature shall be obtained from the **subjectPublicKeyInfo** field of the issuing CA's certificate.

6.1.2 Digital Signature Algorithm (DSA)

X.509 certificates specify both the algorithm used to sign the certificate (in the **signature** field) and the algorithm of the subject's public key (in the **subjectPublicKeyInfo** field). The Travel System CA shall be able to sign certificates and Certificate Revocation Lists (CRLs) using the DSA as specified below. End entities shall be able to sign with the DSS algorithm specified below. Clients shall be able to validate DSS signatures as specified below.

The Digital Signature Algorithm is defined in FIPS 186, The Digital Signature Standard [FIPS186]. It shall be used with the SHA-1 hash algorithm. The ASN.1 object identifier used to identify the DSS algorithm shall be:

```
dsaWithSHA-1 OBJECT IDENTIFIER  ::=  {
        iso(1) identified-organization(3) oiw(14) secsig(3)
        algorithm(2) 27 }
```

The abstract syntax indicating use of the DSA includes optional parameters. These parameters are commonly referred to as p, q, and g. The **AlgorithmIdentifier** within **subjectPublicKeyInfo** is the only place within a certificate where these parameters shall be present. If the DSA parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the subject certificate using DSA, then the certificate issuer's DSA parameters apply to the subject's DSA key. If the DSA algorithm parameters are absent from the **subjectPublicKeyInfo AlgorithmIdentifier** and the CA signed the certificate using a signature algorithm other than DSA, then clients shall not validate the certificate. The parameters are included using the following ASN.1 structure:

```
Dss-Parms  ::=  SEQUENCE  {
        p       INTEGER,
        q       INTEGER,
        g       INTEGER  }
```

When signing, the DSA algorithm generates two values. These values are commonly referred to as r and s. To easily transfer these two values as one signature, they shall be ASN.1 encoded using the following ASN.1 structure:

```
Dss-Sig-Value  ::=  SEQUENCE  {
        r       INTEGER,
        s       INTEGER  }

        algorithm(2) sha1(26) }
```

6.1.3 Certificate Extensions

A set of standardized extensions has been developed and is specified in an amendment to X.509 [DAM]. Extensions have three components: extension name, criticality flag, and extension value. As specified in the amendment to X.509 [DAM], clients shall not validate certificates that contain an extension with the criticality flag set, unless the client can process that extension.

The standardized extensions that have been defined may be divided into four categories: key and policy information; subject and issuer attributes; certification path constraints; and CRL identification extensions.

6.1.3.1 Key and Policy Information

These extensions provide information to identify a particular public key and certificate. They can be used to identify a particular public key/certificate for a CA which has several certificates. This may help a client to find the particular CA certificate needed to establish a certification path. These extensions may restrict the purposes for which a key may be used, and provide information in CA certificates about equivalent policies.

Authority Key Identifier

The **authorityKeyIdentifier** extension provides a means of identifying the particular public key used to sign a certificate. The identification can be based on either the key identifier or on the issuer name and serial number. The key identifier method shall be used in certificates conforming to this interoperability specification. This extension is used where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). The Travel System CA shall be capable of generating this extension, and clients shall be capable of finding and validating certification paths where the issuing CA has several digital signature keys. Clients shall be able to process either the key identifier or the certificate issuer plus certificate serial number form of key identifier if they use this extension to find certification paths.

Subject Key Identifier

This field enables differentiation of keys held by a subject. This field shall be included in every certificate issued. The 160-bit SHA-1 hash of the subject public key shall be used as the **keyidentifier** in the **subjectKeyIdentifier** field used in certificates conforming to this interoperability specification. The hash shall be calculated over the value (excluding tag and length) of the subject public key field in the certificate. This extension shall be non-critical.

Key Usage

The **keyUsage** extension defines restrictions on the use of the key contained in the certificate based on policy and/or usage (e.g., signature, encryption). The Travel System CA shall support the generation of this extension and clients shall be capable of processing it. This extension shall be set to critical.

Private Key Usage Period

The **privateKeyUsagePeriod** extension applies only to digital signature keys. A signature on a document that purports to be dated outside the private key usage period is not valid. This extension shall not be included in certificates and clients are not required to check this field.

Certificate Policies

The **certificatePolicies** extension contains one or more object identifiers (OIDs). Each OID indicates a policy under which the certificate has been issued. The Travel System CA shall be able to generate certificates with one or more instances of **policyIdentifier**. Use of the **policyQualifiers** subfield is optional; if used, it shall be used only to provide information about obtaining CA policy statements.

Clients shall be capable of processing **policyIdentifier** fields against a list of acceptable policies. Clients shall compare the policy identifier(s) in the certificate to that list. Clients shall validate the certification path only if at least one of the policy OIDs in the **certificatePolicies** field in each certificate in the path matches one of the policies in the list of acceptable policies.

Policy Mapping

This non-critical extension is used in CA certificates. It lists the OIDs of equivalent policies. The **issuerDomainPolicy** is considered to be equivalent to the **subjectDomainPolicy**. The Travel System CA shall be capable of generating the **policyMappings** extension. Clients shall be capable of processing this extension.

6.1.3.2 Certificate Subject and Issuer Attributes

The **subjectAltName**, **issuerAltName** and **subjectDirectoryAttributes** are all non-critical extensions. They provide additional information about other names and attributes of the subject and issuer.

Alternative Name

The **subjectAltName** and **issuerAltName** extensions allow additional identities to be bound to the subject and issuer of the certificate. Defined options include an RFC822 [RFC 822] name (electronic mail address), a DNS name, and a URL. Multiple instances may be included. Whenever such identities are to be bound in a certificate, the **subjectAltName** or **issuerAltName** fields shall be used. The Travel System CA shall support the inclusion of alternative names in certificates.

Although X.509 allows null certificate **subject** or **issuer** field accompanied by a critical **subjectAltName** or **issuerAltName** , the Travel System shall not support such certificates. Clients are not required to process any alternative name format nor need they be able to process certificates with null **subject** or **issuer** fields.

The **subjectAltName** and **issuerAltName** extensions are always non-critical in certificates issued by the Travel System CA. A Travel System client which recognizes these extensions need not be able to process all the alternatives of the choice. If the alternative used is not supported by the client, the extension field is ignored.

Subject Directory Attributes

The **subjectDirectoryAttributes** extension may hold any information about the subject where that information has a defined X.500 Directory attribute. This extension is always non-critical. Implementation and use of this extension is optional.

6.1.3.3 Certification Path Constraints

The **basicConstraints**, **nameConstraints** and **policyConstraints** all apply restrictions to valid certification paths.

Basic Constraints

The **basicConstraints** extension tells whether the subject of the certificate is a CA through the **cA** component and the lengths of certification paths through the **pathLenConstraint** component. The Travel System CA shall support the generation of the **basicConstraints** extension in certificates and clients shall be capable of processing it. The

pathLenConstraint extension is meaningful only if **cA** is set to TRUE. The **cA** field shall be included in all certificates. The **basicConstraints** extension shall be marked as critical.

Name Constraints

The **nameConstraints** field applies only to CA certificates. It indicates a name space in which all subsequent certificates in a certification path must be located. The Travel System CA shall be capable of including this field in certificates and clients shall be capable of processing it. If used, it shall be critical.

Policy Constraints

The **policyConstraints** extension serves two functions. It can require that a specific policy apply to all or to a portion of the CA path. It can also inhibit policy mapping for all or a selected portion of the certification path. The Travel System CA shall be capable of supporting the issuance of certificates with this extension, and clients shall be capable of processing this extension. If used, it shall be critical.

6.1.3.4 CRL Identification Extensions

These extensions include information in a certificate about where to obtain the Certificate Revocation List (CRL) that applies to that certificate. They facilitate the division of a CA's potentially large CRL into several shorter CRLs, by identifying in the certificate which CRL applies to a certificate and give the name of the CRL issuer (which may be a CA other than the CA that issued the certificate).

CRL Distribution Points

The **cRLDistributionPoints** extension identifies the CRL distribution point or points to which a clients should refer to ascertain if a certificate has been revoked. This field has three component fields: **distributionPoint**, **reasons** and **cRLIssuer**.

- The **distributionPoint** component identifies the location from which the CRL can be obtained. If this field is absent, the CRL distribution point name defaults to the issuer name. This extension provides a mechanism to divide the CRL into manageable pieces if the CA has a large constituency.
- The **reasons** component identifies the reasons for revocation covered by the CRL issued by the corresponding **distributionPoint**. If the **reasons** component is absent, the corresponding **distributionPoint** distributes a CRL which will contain an entry for this certificate, if it has been revoked for any reason. A **reasons** value of **certificateHold** shall not be used. Clients are not required to process the **reasons** component.
- The **cRLIssuer** component identifies the authority that issues and signs the CRL. If this component is absent, the CRL issuer name defaults to the certificate issuer name. One use for this component is to allow the construction of consolidated CRLs, that include certificates issued by more than one CA.

The Travel System CA shall be able to generate the **cRLDistributionPoints** extension and all its components in certificates. Clients shall be able to use distribution point CRLs and validate CRLs where the **cRLIssuer** component is used. See 6.2 below for a further discussion of distribution points.

6.1.3.5 Summary of Certificate Extension Use

Table 6-1 summarizes the standardized certificate extensions, while Table 6-2 summarizes the use of standardized extensions for certificates and clients by the Travel System.

Table 6-1 Summary of Standardized Certificate Extensions

Extension	Used By	Use	Critical
Key and Policy Information			
keyIdentifier	all	identifies the key used to sign this certificate (the signing CA may have several keys)	No
authorityKeyIdentifier	all	unique with respect to authority.	
authorityCertIssuer	all	identifies issuing authority of CA's certificate; alternative to key identifier	
authorityCertSerialNumber	all	used with authorityCertIssuer	
subjectKeyIdentifier	all	enables differentiation of different keys for same subject. Must be unique for subject.	No
keyUsage	all	defines allowed purposes for use of key (e.g., digital signature, key agreement...)	Yes*
privateKeyUsagePeriod	all	digital signature keys only. Signatures on documents that purport to be dated outside the period are invalid.	Yes*
certificatePolicies	all	policy identifiers and qualifiers that identify and qualify policies applying to the certificate	No*
policyIdentifiers	all	the OID of a policy.	
policyQualifiers	all	more information about the policy	
policyMappings	CA	indicates equivalent policies	No
Certificate Subject and Issuer Attributes			
subjectAltName	all	used to list alternative names (e.g., rfc822 name, X.400 address, IP address...)	No*
issuerAltName	all	used to list alternative names	No*
subjectDirectoryAttributes	all	any attributes (e.g., supported algorithms)	No
Certification Path Constraints			
basicConstraints	all	constraints on subject's role & path lengths	Yes*
cA	all	distinguish CA from end entity cert.	
pathLenConstraint	CA	max. number of following CAs in cert. path; 0 indicates that CA only issues end entity certs.	
nameConstraints	CA	limits subsequent CA cert. Name space.	Yes*
permittedSubtrees	CA	names outside indicated subtrees are forbidden	
excludedSubtrees	CA	indicates disallowed subtrees	
policyConstraints	all	constrains certs. Issued by subsequent CAs	Yes*
policySet	all	those policies to which constraints apply	
requireExplicitPolicy	all	All certs. following in the cert. path must contain an acceptable policy identifier	
inhibitPolicyMapping	all	prevent policy mapping in following certs.	
CRL Identification			
crlDistributionPoints	all	divides long CRL into shorter lists	No*
distributionPoint	all	location from which CRL can be obtained	
reasons	all	reasons for cert. inclusion in CRL	
cRLIssuer	all	name of component that issues CRL.	

NOTES:

* Standard allows either critical or noncritical.

Table 6-2 Use of Standardized Certificates by the Travel System

Extension	Certificate	Client
Key and Policy Information		
authorityKeyIdentifier		
authorityKeyIdentifier	to be included in all certs issued: a random number large enough to generally be globally unique	optional - may be used to help find cert. paths where issuer has multiple certs. (1)
authorityCertIssuer	not used	optional - used to find cert. paths where issuer has multiple certs. (1)
authorityCertSerialNumber	not used	
subjectKeyIdentifier	to be included in all certs issued: a random number large enough to generally be globally unique	supported: used with CRLs to identify revoked certificates.
keyUsage	supported	supported
privateKeyUsagePeriod	supported	not used
certificatePolicies		
policyIdentifiers	supported	supported; compared during cert. path validation with a list of acceptable policies
policyQualifiers	not used	supported
policyMappings	supported	supported
Certificate Subject and Issuer Attributes		
subjectAltName	supported	not used
issuerAltName	supported	not used
subjectDirectoryAttributes	not used	not used
Certification Path Constraints		
basicConstraints		
cA	used in all certificates	supported
pathLenConstraint	supported	supported
nameConstraints		
permittedSubtrees	supported	supported
excludedSubtrees	supported	supported
policyConstraints		
policySet	supported	supported
requireExplicitPolicy	supported	supported
inhibitPolicyMapping	supported	supported
CRL Identification		
cRLDistributionPoints		
distributionPoint	supported	supported
reasons	supported	supported
cRLIssuer	supported	supported

NOTES:

For Certificates, “supported” means that the CA shall be able to issue certificates that contain this extension. For clients, “supported” means that clients shall be capable of processing this extension.

(1) Clients shall be capable of finding certification paths where CAs have multiple certificates, whether or not they use this extension to do so.

6.2 Certificate Revocation List (CRL)

Certificate Revocation Lists (CRL) are used to list unexpired certificates that have been revoked. Certificates may be revoked for a variety of reasons, ranging from routine administrative revocations, (when the certificate's subject leaves the issuing organization, or when responsibilities and certificate attributes change), to situations where the private key is compromised. The X.509 v2 certificate revocation list format is augmented by several optional extensions, similar in concept to those defined for certificates. The Travel System CA shall be able to generate X.509 v2 CRLs as specified below, and clients shall be capable of processing them when validating certification paths. The X.509 v2 CRL includes the following:

Version
Issuer Signature Algorithm
Issuer Distinguished Name
This Update
Next Update
Revoked Certificates, a sequence of one or more of the following sequence:
    Certificate Serial Number
    Revocation Date
    CRL Entry Extensions (optional)
CRL Extensions (optional)
Issuer's Signature on all the above listed fields

6.2.1 CRL Fields

The X.509 v2 CRL ASN.1 syntax is given in Appendix B. For signature calculation, the data that is to be signed is ASN.1 DER encoded. ASN.1 DER encoding is a tag, length, value encoding system for each element.

The following items describe the use of the X.509 v2 CRL.

Version

This field describes the version of the encoded CRL. The value of this field shall be 1, indicating a v2 CRL.

Signature

The **signature** field contains the algorithm identifier for the algorithm used to sign the CRL. The contents are identical to the contents of the certificate **signature** field. Refer to Signature in Section 6.1 for information about this field. CRLs issued by the Travel System CA are signed with the DSA algorithm. Refer to Section 6.1.2 for the signature algorithms. The **parameters** subfield of the CRL **signature** field shall not be used to pass DSA parameters; rather DSA parameters shall be obtained from the **subjectPublicKeyInfo** field of the certificate of the Travel System CA.

Issuer Name

The **issuer** field provides a globally unique identifier of the CA signing the CRL. The issuer name is an X.500 distinguished name. CRL issuer names with empty sequences are not supported by the Travel System CA or its clients.

This Update

The **thisUpdate** field indicates the date of the CRL. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see Section 6.1.1 above).

Next Update

The **nextUpdate** field indicates the date by which the next CRL will be issued. The next CRL could be issued before the indicated date, but it will not be issued any later than the indicated date. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see Section 6.1.1 above).

Revoked Certificates

The **revokedCertificates** field is a list of the certificates that have been revoked. Each revoked certificate listed contains:

- the certificate serial number, stated in the **userCertificate** field. This element contains the value of **serialNumber** of the revoked certificate. This must be used in conjunction with the name of the issuing CA to identify an unexpired certificate that has been revoked.
- the **revocationDate** field that contains the date of the revocation in **UTCTime** format. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see Section 6.1.1 above).
- optional CRL entry extensions, that are specified in Section 6.2.3 below. The CRL entry extensions may give the reason that the certificate was revoked, state the date that the invalidity is believed to have occurred, and may state the name of the CA that issued the revoked certificate, which may be a different CA from the CA issuing the CRL. Note that the CA that issued the CRL is assumed to be the CA that issued the revoked certificate unless the **certificateIssuer** CRL entry extension is included.

6.2.2 CRL Extensions

The extensions defined by ISO/ITU for X.509 v2 CRLs provide methods for associating additional attributes with entire CRLs. Each CRL extension may be designated as critical or non-critical. A CRL validation shall fail if a client encounters a critical extension that it cannot process.

This section describes CRL extensions that shall be supported. A CRL extension is supported when: the CA is able to generate the extensions in a CRL and the clients are able to process the extension.

Authority Key Identifier

The **authorityKeyIdentifier** is a non-critical CRL extension that identifies the CA's key used to sign the CRL. This extension is useful when a CA uses more than one key; it allows distinct keys differentiated (e.g., as key updating occurs). The identification can be based on either the key identifier or on the issuer name and serial number. The key identifier method shall be used, and the **keyIdentifier** shall be generated for all CRLs. This extension is useful where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). This extension shall be included in all CRLs, and clients shall be able to find and validate CRL certification paths where the issuing CA has multiple signing keys. Clients shall be able to process either the key identifier or the certificate issuer plus serial number form of **authorityKeyIdentifier** if they use this extension to find certification paths.

Issuer Alternative Name

The **issuerAltName** is a non-critical CRL extension that provides alternative CA names. Multiple instances may be included. Whenever such alternative names are included in a CRL, the issuer alternative name field shall be used. Implementations which recognize this extension need not be able to process all the alternative name formats. Unrecognized alternative name formats may be ignored by the Travel System CA or its clients. The Travel System CA shall be capable of generating this extension in CRLs, however clients are not required to process it.

CRL Number

The **cRLNumber** field is a non-critical CRL extension which conveys a monotonically increasing sequence number for each CRL issued by a given CA through a specific CA directory entry or CRL distribution point. This extension allows certificate users to easily determine when a particular CRL supersedes another CRL. This extension shall be included in CRLs.

Issuing Distribution Point

The **issuingDistributionPoint** field is a critical CRL extension that identifies the CRL distribution point for this particular CRL. A distribution point is a directory entry that may be used to retrieve a CRL, and that may differ from the directory entry of the issuing CA. The CRL is signed by the CA's key. CRL distribution points do not have their own key pairs.

In addition, the **issuingDistributionPoint** field specifies CRLs that may contain only end entity certificates, or only CA certificates, or only certificates that have been revoked for a particular reason. Finally, this extension can identify an “indirect CRL,” that is a CRL that is issued by a different CA than the CA(s) that issued the revoked certificate. It contains the following components:

- **distributionPoint**, which gives the name of the distribution point name. If used, **distributionPoint** shall be an X.500 distinguished name;

- **onlyContainsUserCerts**, a Boolean value that indicates that the CRL contains only end entity certificates;
- **onlyContainsCACerts**, a Boolean value that indicates that the CRL contains only CA certificates;
- **onlySomeReasons**, a **ReasonsFlag** bit string that indicates the reasons for which certificates are listed in the CRL. Only the following reason flags shall be included in CRLs:
  - **keyCompromise** shall be used to indicate compromise or suspected compromise;
  - **cACompromise** shall be used to indicate that the certificate has been revoked because of a CA key compromise. It shall only be used to revoke CA certificates;
  - **affiliationChanged** shall be used to indicate that the certificate was revoked because of a change of affiliation of the certificate subject;
  - **superseded** shall be used to indicate that the certificate has been superseded;
  - **cessationOfOperation** shall be used to indicate that the certificate is no longer needed for the purpose for which it was issued, but there is no reason to suspect that the private key has been compromised.
- **indirectCRL**, a Boolean value that indicates that this is an indirect CRL.

Clients shall be able to process this field. CRLs shall not include an **onlySomeReasons** entry that indicates **certificateHold**.

Delta CRL Indicator

The **deltaCRLIndicator** is a critical CRL extension that identifies a delta-CRL. The use of delta-CRLs can significantly improve processing time for applications which store revocation information in a format other than the CRL structure. This allows changes to be added to the local database while ignoring unchanged information that is already in the local database.

The value of **BaseCRLNumber** identifies the CRL number of the base CRL that was used as the starting point in the generation of this delta-CRL. The delta-CRL contains the changes between the base CRL and the current CRL. A delta-CRL is not issued by itself; if a delta-CRL is issued a complete current CRL is also issued. It is the decision of a CA as to whether to provide delta-CRLs. A delta-CRL shall not be issued without a corresponding base CRL. The value of CRL number for both the delta-CRL and the corresponding base CRL shall be identical.

A client constructing a locally held CRL from delta-CRLs shall consider the constructed CRL incomplete and unusable if the CRL number of the received delta-CRL is more that one greater that the CRL number of the delta-CRL last processed. Support of delta-CRLs by the Travel System CA and clients is optional.

Summary of CRL Extension Use

Table 6-3 summarizes the standardized CRL extensions, while Table 6-4 summarizes the use of the standardized CRL extensions for the Travel System.

6.2.3 CRL Entry Extensions

The CRL entry extensions defined for X.509 v2 CRLs provide methods for associating additional attributes with CRL entries. Each extension in a CRL entry is designated as critical or non-critical. A CRL validation shall fail if it encounters a critical CRL entry extension which it does not know how to process. However, an unrecognized non-critical CRL entry extension may be ignored.

Table 6-3 Summary of CRL Extensions

Extension	Use	Critical
authorityKeyIdentifier	identifies the CA key used to sign CRL.	No
keyIdentifier	unique key identifier; alternative to **certIssuer** & **authorityCertSerialNumber**	
certIssuer	name of CA's cert. issuer	
authorityCertSerialNumber	used with **certIssuer**; combination must be unique	
issuerAltName	alternate name of CRL issuer	No*
cRLNumber	sequence number for CRL	No
issuingDistributionPoint	name of CRL distribution point; also gives reasons for revocations contained in CRL.	Yes
deltaCRLIndicator	indicates delta CRL (lists certificates. revoked since last full CRL) & gives sequence number	Yes

NOTES:

* Standard allows either critical or noncritical. Indication is for use in the DoE Travel System.

Table 6-4 Summary of CRL Extensions and their use in the Travel System

Extension	CRL	Clients
authorityKeyIdentifier		
keyIdentifier	included in all CRLs issued	optional - used to help find correct CA certificate to validate CRL (1)
certIssuer	not generated	optional - issuer/serial number pair used to help find correct authority certificate to validate CRL (1)
certSerialNumber	not generated	
issuerAltName	supported	optional
cRLNumber	supported: included in all CRLs	optional
issuingDistributionPoint	supported	supported
deltaCRLIndicator	optional	optional

NOTES:

- For CRLs, "supported" means that the CA is capable of issuing CRLs that contain this extension.
- For Clients, "supported" means that the client is capable of processing this extension in CRLs.

(1) Clients shall be capable of finding the certificate used to sign a CRL, when the CA has multiple certificates, and the certificates are accessible in the appropriate directory, whether or not they use this extension to do so, and whether or not the CRL contains this extension.

Reason Code

The **reasonCode** is a non-critical CRL entry extension that identifies the reason for the certificate revocation. The Travel System CA shall be capable of generating this extension in CRL entries. Processing of the **reasonCode** extension by clients is optional, that is clients shall not validate a certificate if any certificate in the certification path is listed in a current CRL, regardless of the **reasonCode,** and need not provide operator information about the reason for failure. The following enumerated **reasonCode** values are defined:

- **unspecified**; this value shall not be used;
- **keyCompromise** indicates compromise or suspected compromise;
- **cACompromise** indicates that the certificate has been revoked because of a CA key compromise. It shall only be used to revoke CA certificates;
- **affiliationChanged** indicates that the certificate was revoked because of a change of affiliation of the certificate subject;
- **superseded** indicates that the certificate has been replaced by a more recent certificate;

- **cessationOfOperation** indicates that the certificate is no longer needed for the purpose for which it was issued, but there is no reason to suspect that the private key has been compromised.
- **certificateHold** shall not be used. When clients process a certificate that is listed in a CRL with a **reasonCode** of **certificateHold**, they shall fail to validate the certification path.
- **removeFromCRL**, which is used only with delta-CRLs and indicates that an existing CRL entry should be removed.

Expiration Date

The **expirationDate** is a non-critical CRL entry extension that indicates the expiration of a hold entry in a CRL. This extension shall not be used in CRLs or by clients.

Instruction Code

The **instructionCode** is a non-critical CRL entry extension that provides a registered instruction identifier which indicates the action to be taken after encountering a certificate that has been placed on hold. This extension shall not be used in CRLs.

Invalidity Date

The **invalidityDate** is a non-critical CRL entry extension that provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry. The revocation date in the CRL entry specifies the date that the CA revoked the certificate. Whenever this information is available, CAs are encouraged to share it with CRL users. The Travel System CA shall be capable of generating this extension in CRLs. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see Section 6.1.1 above).

Certificate Issuer

The **certificateIssuer** CRL entry extension is used with an indirect CRL (a CRL that has the **indirectCRL** indicator set in its **issuingDistributionPoint** extension). If this extension is not present in the first entry of an indirect CRL, the certificate issuer defaults to the CRL issuer. In subsequent entries in an indirect CRL, when the **certificateIssuer** extension is not present, the certificate issuer is the same as the issuer of the preceding CRL entry.

Summary of CRL Entry Extension Use

Table 6-5 summarizes the CRL entry extensions while Table 6-6 summarizes the use of CRL entry extensions for the DoE Travel System.

Table 6-5 Summary of CRL Entry Extensions

Extension	Use	Critical
reasonCode	identifies the reason for the revocation of this certificate	No
instructionCode	used with **certificateHold reasonCode**; indicates action to be taken when encountering a held certificate	No
invalidityDate	date certificate became invalid	No
certificateIssuer	Issuer of revoked certificate in an indirect CRL	Yes

Table 6-6 Summary of CRL Entry Extensions Use in the DoE Travel System

Extension	CRL	Clients
reasonCode	supported; included for all entries; **certificateHold** not used	optional - may be used to provide information about validation failure
instructionCode	not used	optional
invalidityDate	supported	optional - may be used to provide information about validation failure
certificateIssuer	supported	supported

NOTES

For CRLs, "supported" means the CA is capable of issuing CRLs that contain this CRL entry extension. For clients, "supported" means that the client is capable of processing this entry extension in CRLs.

6.3 Certification Path Validation

The procedure specified in Section 12.4.3 of the DAM [DAM], Certification path processing procedure, shall be adopted by clients with the following modifications:

- The default value for input (d) *initial-explicit-policy* indicator shall indicate a specific acceptable policy identifier that must appear in the certificate policies extension of every certificate in the certification path;
- The default value for input (e) *initial-policy-mapping-inhibit* shall be TRUE, i.e., policy mapping need not be allowed in complying implementations;
- The diagnostic codes provided upon failure of certification path validation in output (b) shall be provided by the contractor.
- Information provided upon success of a path validation in output (c) is optional for complying implementations;
- Policy mapping information in output (e) is optional;
- Support for *pending-constraints* state information is optional.

6.4 Transaction Message Formats

This section presents a set of message formats to support the minimal set of PKI transactions. Systems that implement these transactions shall support these message formats, generating and recognizing them as appropriate. The message formats are specified in ASN.1; messages shall be encoded and transmitted using the Distinguished Encoding Rules (DER).

6.4.1 Overall PKI Message Components

PKI Message

Each message has three components

```
PKIMessage ::= SEQUENCE {
    header          PKIHeader,
    body            PKIBody,
    protection      PKIProtection
 }
```

PKI Message Header

All PKI messages require some header information for addressing and transaction identification. Some of this information will also be present in a transport specific envelope, however, if the PKI message is signed then this information is also protected (i.e. we make no assumption about secure transport).

The following data structure is used to contain this information:

```
 PKIHeader ::= SEQUENCE {
        pvno                INTEGER     { fpki-version1 (0) },
        messageType         MessageType,
        transactionID       [1] OCTET STRING     OPTIONAL,
                -- identifies the transaction, i.e. this will be the
                -- same in corresponding request, response and
                -- confirmation messages
        messageID           [2] OCTET STRING     OPTIONAL,
                -- identifies this message uniquely (if needed)
        senderNonce         [3] OCTET STRING     OPTIONAL,
        recipNonce          [4] OCTET STRING     OPTIONAL,
                -- nonces used to provide replay protection, senderNonce
                -- is inserted by the creator of this message; recipNonce
                -- is a nonce previously inserted in a related message by
                -- the intended recipient of this message
        messageTime         UTCTime              OPTIONAL,
                -- time of production of this message
        sender                          GeneralName,
                -- identifies the sender for addressing purposes
        recipient           GeneralName,
                -- identifies the intended recipient for
                -- addressing purposes
        protectionAlg       AlgorithmIdentifier     OPTIONAL,
                 -- to include the alg. in e.g. signature calculation
        freeText            CHOICE {
```

```
                                IA5String,
                            BNPString }          OPTIONAL
        -- this may be used to indicate context specific
        -- instructions - this field is intended for human
        -- consumption
}

MessageType ::= INTEGER {
    CertReq             (2), -- message asking for a cert.
    CertRep             (3), -- response to above
    KeyUpdReq           (4), -- msg. asking for cert for new public  key
    KeyUpdRep           (5)  -- response to above
    RevReq              (8), -- message asking for revocation
    RevRep              (9), -- response to above
    PKIConfirm          (16), -- used for confirmation (ACK)
    PKIEnvelope         (18) -- used for doubly signed messages
}
```

The **transactionID** field within the message header is required so that the recipient of a response message can correlate this with the request. In the case of an ORA there may be many request "outstanding" at a given moment. The value of this field should be unique from the sender's perspective in order to be useful.

The **messageTime** field indicates the time the message was generated. The **UTCTime** (Coordinated Universal Time) value included in this field shall follow the rules for the certificate **validity** field (see Section 6.1.1 above).

The **sender** and **recipient** fields within the message header are defined as **GeneralName**. Systems are required to support X.500 distinguished names and RFC 822 (Internet electronic mail) names.

The **protectionAlg** is required for all signed messages. The **messageID**, **senderNonce**, and **recipNonce** fields are not required.

PKI Message Body

```
PKIBody ::= CHOICE {
    -- message specific body elements
        [2]     CertReqContent,
        [3]     CertRepContent,
        [4]     KeyUpdReqContent,
        [5]     KeyUpdRepContent,
        [8]     RevReqContent,
        [9]     RevRepContent,
        [16]    PKIConfirmContent,
        [18]    InnerMessage
}
```

PKI Message Protection

Some PKI messages will be protected for integrity. In that case the following structure is used:

```
PKIProtection ::= SEQUENCE {
    alg          AlgorithmIdentifier    OPTIONAL,
        -- when both are present this should be the same as the
        -- protectionAlg field of the PKIHeader
    protectionBits   BIT STRING
}
```

The input to the calculation of the **protectionBits** is the DER encoding of the following data structure:

```
ProtectedPart ::= SEQUENCE {
        PKIHeader,
        PKIBody}
```

Note that this is equivalent to the ASN.1 notation:

```
PKIProtection ::= SIGNATURE SEQUENCE {
        PKIHeader,
        PKIBody}
```

In some cases, such as key update, it may be necessary to attach multiple signatures. In this case, signatures are applied iteratively - each signed message becomes the body of a **PKIMessage** of type **PKIEnvelope**, until all signatures have been applied.

6.4.2 Common Data Structures

The following data types are common to several message formats.

Certificate Templates

In various PKI management messages, the originator may provide certain values to identify an existing certificate or request certain values be used in the generation of a certificate. The **CertTemplate** structure allows entities to indicate those values. **CertTemplate** includes all the same information as a certificate.

The **CertTemplates** structure is a sequence of **CertTemplate**. This structure permits "batch processing" of requests in a single transaction. Since this may also be performed through a series of transactions, this feature is not required. **CertTemplates** may be considered a sequence of exactly one **CertTemplate** wherever it appears.

```
CertTemplate ::= SEQUENCE {
        version          [0] Version               OPTIONAL,
                -- used to ask for a particular syntax version
        serial           [1] INTEGER               OPTIONAL,
                -- used to ask for a particular serial number
        signingAlg       [2] AlgorithmIdentifier OPTIONAL,
        subject          [3] Name                  OPTIONAL,
        validity         [4] OptionalValidity      OPTIONAL,   -- policy
        issuer           [5] Name                  OPTIONAL,
        publicKey        [6] SubjectPublicKeyInfo  OPTIONAL,  -- required
        issuerUID        [7] UniqueIdentifier      OPTIONAL,   -- not supported
        subjectUID       [8] UniqueIdentifier      OPTIONAL,   -- not supported
        extensions       [9] Extensions            OPTIONAL,
```

```
                -- contains the extensions which the requester
                -- would like in the cert.
}

OptionalValidity ::= SEQUENCE {
        notBefore       [0] UTCTime OPTIONAL,
        notAfter        [1] UTCTime OPTIONAL
}

CertTemplates ::= SEQUENCE OF CertTemplate
```

Status codes for PKI messages

All response messages will include some status information. The following values are defined:

```
PKIStatus ::= INTEGER {
        granted                 (0),
                -- request granted without change
        grantedWithMods         (1),
                -- request granted, with modifications; the  requester
                -- is responsible for ascertaining the differences
        rejection               (2),
                 -- request rejected
        waiting                 (3),
                 -- the request has been received but has not been processed,
                -- an additional response will follow after processing
        revocationWarning       (4),
                 -- this message contains a warning that a revocation is
                 -- imminent
        revocationNotification  (5)
                -- notification that a revocation has occurred
}
```

Failure Information

Responders use the following syntax to provide more information about failure cases.

```
PKIFailureInfo ::= BIT STRING {    -- since we can fail in more than
                        -- one way!
        badAlg                  (0),    -- unrecognized  or unsupported algorithm identifier
        badMessageCheck         (1),    -- integrity check failed (signature did not verify)
        badRequest              (2),    -- transaction not permitted or supported
        badTime                 (3),    -- messageTime field was not sufficiently close
                                        -- to the system time
        badCertId               (4)     -- no certificate could be identified matching the
                                        -- provided criteria
        -- need more failure information
}
```

Protocol Confirmation

This data structure has no content in all cases. Confirmation messages shall carry all the required information in the **PKIHeader**.

```
PKIConfirmContent ::= NULL
```

Certificate Identification

In order to identify particular certificates the **CertId** structure is used.

```
CertId ::= SEQUENCE {
        issuer            GeneralName,
        serialNumber    INTEGER
  }
```

6.4.3 Operation-Specific Data Structures

Registration/Certification Request

Registration/Certification request message (**CertReq**) contains a **CertReqContent** data structure which specifies values for one or more requested certificates.

CertReqContent ::= CertTemplates

The certificate request body shall include the prospective certificate holder's distinguished name and public key in the **subject** and **publicKey** fields.

Registration/Certification Response

A registration response message (**CertRep**) contains a **CertRepContent** structure which has a status value and optionally failure information, a CA public key, subject certificate and encrypted private key.

```
CertRepContent ::= SEQUENCE {
        status          PKIStatus,
        failInfo        [0] PKIFailInfo  OPTIONAL,    -- present if status is rejection
        caPub           [1] OOBCert      OPTIONAL,
        certificate     [2] Certificate  OPTIONAL,    -- present if status is granted or
                                                      -- grantedWithMods
        privateKey      [3] EncPrivKey   OPTIONAL     -- not required
  }
```

Only one of the **failInfo** or **certificate** should be present (depending on the status). For the status values **waiting** none of the optional fields will be present. The status values **revocationWarning** and **revocationNotification** should not appear in this message.

For the DoE Travel System, the **privateKey** field shall not be used. The **caPub** field is not required, and may be ignored if present.

Revocation Request Content

When requesting revocation of a certificate the following data structure is used. The name of the requester is present in the **PKIHeader** structure.

```
RevReqContent ::= SEQUENCE {
     certDetails          CertReqContent,
          -- allows requester to specify as much as they can about
          -- the cert. for which revocation is requested
          -- (e.g. for case serialNumber not available)
     revocationReason ReasonFlags,
          -- from the DAM, so that CA knows which Dist. point to use
```

```
    badSinceDate      UTCTime    OPTIONAL,
        -- indicates best knowledge of sender
    crlEntryDetails   Extensions}
        -- requested crlEntryExtensions
```

ReasonFlags are defined in Appendix B. but are reproduced here for clarity.

```
ReasonFlags ::= BIT STRING {
        unused                  (0),
        keyCompromise           (1),
        caCompromise            (2),
        affiliationChanged      (3),
        superseded              (4),
        cessationOfOperation    (5),
        certificateHold         (6),
        removeFromCRL           (8) }
```

Revocation Response Content

The response to the above message. If produced this is sent to the requester of the revocation. (A separate revocation announcement message may be sent to the subject of the certificate which was requested to be revoked.)

```
RevRepContent ::= SEQUENCE {
        status          PKIStatus,
        failInfo        PKIFailInfo             OPTIONAL,
        revDetails      [0] CertId              OPTIONAL,
                -- identifies the cert for which revocation
                -- was requested
        cRL             [1] CertificateList     OPTIONAL}
                -- the resulting CRL
```

Key update request content

For certificate renewal transactions performed directly between certificate holders and CAs, the following syntax is used

```
KeyUpdReqContent ::= SEQUENCE {
        endEntityName       GeneralName,
        latestCerts         SEQUENCE OF CertId,
        protocolEncKey      [1] SubjectPublicKeyInfo OPTIONAL,
        certTemplates       [2] CertTemplates OPTIONAL
}
```

In the DoE Travel System, **latestCerts** shall be a sequence of exactly one **CertId**. If the **endEntityName** is an X.500 distinguished name, it will correspond to the **subject** field in the certificate. Otherwise, the **endEntityName** field will specify the RFC822 electronic mail address for the end entity and can be used to support communications protocols. The **ProtocolEncKey** field conveys the new public key for the new certificate.[7] The **certTemplates** includes proposed values for any modifications in the certificate fields. If omitted, the only fields that will change are the public key and the validity period.

[7] If omitted, the user is requesting a new certificate with the same key. It is a policy decision if such a request will be honored.

Key Update response content

This is just like any other certification response.

```
KeyUpdRepContent ::= InitRepContent

InitRepContent ::= SEQUENCE {
        referenceNum            INTEGER,
        protocolEncKey          [0] SubjectPublicKeyInfo OPTIONAL,
        certTemplates           CertTemplates
}
```

6.5 PKI Transactions

This section describes PKI specific transactions to request, renew, or revoke certificates; the transactions are based on the working document [PKIX3]. This section also provides a brief description of transactions for accessing the directory service.

6.5.1 Initial Certificate Issuance

An ORA may request that the Travel System CA issue a certificate for an end entity. This transaction is performed in three steps. In the first step, the end entity provides a public key to the ORA in a signed message. In the second step, the ORA requests a certificate from the CA in a signed message. The CA replies to the ORA with a signed message containing either a certificate or an error message.

Certificate Request from an End Entity to the ORA

The end entity creates a **PKIMessage** of type **CertReq**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **CertReq**;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the end entity, or null;
- **recipient** is the distinguished name of the ORA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **CertTemplate**. For the DoE Travel System, **CertReqContent** is a sequence of one **CertTemplate**. At a minimum, the **CertTemplate** will include the **publickey** field, which provides the public key for the new certificate.

The **PKIProtection** field contains the end entity's signature, calculated on the DER encoded sequence of the header and body with private key material corresponding to the public key in the **publickey** field.

Certificate Request from ORA to CA

The ORA creates a **PKIMessage** of type **CertReq**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **CertReq**;
- **transactionID** is an integer unique to this transaction for this ORA;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the ORA;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The message body is **CertReqContent**, which is a sequence of one or more **CertTemplate**. For the DoE Travel System, **CertReqContent** is a sequence of one **CertTemplate**. The **CertTemplate** will include the following information:

- **version** is v3 (which is indicated by a value of 2);
- **serial** is used as described below;
- **publickey** provides the public key for the new certificate; and
- **extensions** specifies, at a minimum, the certificate policy OID to be associated with the certificate.

The **serial** field may be set to zero, indicating that the end entity has not previously held a certificate issued by this CA. A non-zero value for **serial** is the serial number for an unexpired, unrevoked certificate issued to this entity by **recipient.**

The following information may be included in the **CertTemplate**:

- **signingAlg** specifies the preferred signature algorithm;
- **subject** is present if and only if serial equals zero, and specifies the distinguished name for the prospective certificate holder;

The request shall not include the following information:

- **issuerUID**; and
- **subjectUID**.

The **PKIProtection** field contains the ORA's signature, calculated on the DER encoded sequence of the header and body.

Certificate Response from CA to ORA

The CA will return a **CertRep** message to the ORA.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **CertRep**;
- **transactionID** is the same as the transactionID field in the CertReq message;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the ORA; and

- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the X.509 version 3 certificate;

The **failInfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failinfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
  - **badMessageCheck** indicates that the signature in the **PKIProtection** field was checked but did not match;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; and
  - **badCertId** indicates that no certificate could be identified matching the non-zero **serial** field, or that the corresponding certificate was revoked or expired.

The **certificate** field may not be present if **status** is **rejected**. If present, the certificate shall conform to the profile presented in Section 6.1.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

6.5.2 Certificate Renewal

An entity that is a current certificate holder may request issuance of a new certificate directly from the CA that issued the current certificate. The requesting entity creates a PKI **KeyUpdReq** message requesting a certificate and signs it with the private key corresponding to the public key in the certificate request. The entity then encapsulates the signed message in a **PKIEnvelope** message which it signs with the private key corresponding to the entity's unexpired, unrevoked certificate.

The Travel System CA shall return a **KeyUpdRep** message to the certificate holder. This message will contain the certificate or a reason code for the transaction failure.

Certificate Renewal Request from Certificate Holder to CA

The certificate holder creates a **PKIMessage** of type **KeyUpdReq**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **KeyUpdReq**;

- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the certificate holder;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is **KeyUpdReqContent**, which is a sequence of **endEntityName**, **latestCerts**, **ProtocolEncKey**, and **CertTemplates**. For the DoE Travel System, **latestCerts** is a sequence of one **CertId** and **CertTemplates** is a sequence of one **CertTemplate**. The **CertId** shall be present and shall identify an unexpired unrevoked certificate issued by the **recipient** identified in the message header.

If present, the **CertTemplate** may include the following information:

- **version** of the certificate to be issued;
- **publickey** provides the public key for the new certificate.
- **signingAlg** specifies the preferred signature algorithm.

The **CertTemplate** shall not include the following information:

- **issuerUID**; and
- **subjectUID**.

The **PKIProtection** field contains a signature generated with the private key associated with the public key specified in the **CertTemplate** and calculated on the DER encoded sequence of the header and body.

This entire message becomes the body of a **PKIMessage** of type **PKIEnvelope**, and is signed with the private key associated with the public key in **ProtocolEncKey**.

Envelope for Renewal Request

The certificate holder now creates a **PKIMessage** of type **PKIEnvelope**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **PKIEnvelope**
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the certificate holder;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is the **PKIMessage** of type **KeyUpdReq** constructed above.

The **PKIProtection** field contains a signature generated using the private key associated with the current unexpired, unrevoked certificate and calculated upon the DER encoded sequence of the header and body.

Certificate Renewal Response from CA to Certificate Holder

The CA will return a **KeyUpdRep** message to the certificate holder.

The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **CertRep**;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the ORA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

If a **transactionID** was supplied in **CertReq** message, the header of the response will include the same **transactionID**.

The **PKIBody** is **CertRepContent**. If the CA issued a certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **certificate** will contain the new X.509 version 3 certificate;

The **failinfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failinfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate the signature because the algorithm identifier is unrecognized or unsupported;
  - **badMessageCheck** indicates that the signature in the **PKIProtection** field was checked but did not match;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; and
  - **badCertId** indicates that no certificate could be identified matching the non-zero **serial** field.

The **certificate** field may not be present if **status** is **rejected**. If present, the certificate shall conform to the profile presented in Section 6.1.

The **PKIProtection** field contains the CA's signature, calculated on the DER encoded sequence of the header and body.

6.5.3 Request Revocation

Certificate holders may request revocation of their own certificates. To perform this function the certificate holder generates a **RevReq** message, signs it with the certificate to be revoked, and sends it to the CA. The **RevReq** message shall include, at a minimum, the

certificate serial number in the serial field of **certDetails** and a revocation reason code in the revocationReason field. The CA responds with a **RevRep** message.

ORAs may request revocation of a certificate issued to an entity on behalf of the certificate holder or the certificate holder's organization. To perform this function, the ORA generates a **RevReq** message, signs it with the ORA's private key, and sends it to the CA. The ORA shall generate a pseudo-random number and shall place it in the transactionID field. The **RevReq** message shall include, at a minimum, the certificate serial number in the serial field of **certDetails** and a revocation reason code in the **revocationReason** field.

The CA will respond to the revocation requester with a **RevRep** message. If the RevReq message includes a **transactionID** , the CA shall include its contents as the **transactionID** in the **RevRep** message. The **RevRep** message shall contain, at a minimum, provide the status of the revocation request in the status field and identify the certificate for which revocation is requested in the **revDetails** field.

Revocation Request from ORA or Certificate Holder to CA

The ORA or the certificate holder creates a **PKIMessage** of type **RevReq**. The **PKIHeader** includes the following information:

- **pvno** is zero;
- **messageType** is **RevReq**;
- **transactionID** is an integer unique to this transaction for this ORA or any integer for the end entity;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the ORA or the certificate holder;
- **recipient** is the distinguished name of the CA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is **RevReqContent**, which is a sequence of **CertDetails**, reason flags, and date and time of compromise or loss. **CertDetails** is defined as a sequence of **CertTemplate**. For the DoE Travel System, **CertDetails** is a sequence of one **CertTemplate**, which will include the following information:

- **serial**, which contains the serial number of the certificate; and
- **issuer**, which contains the distinguished name of the certificate issuer.

or

- **subject**, which contains the distinguished name of the certificate holder; and
- **issuer**, which contains the distinguished name of the certificate issuer.

The **RevDetails** shall also include a reason code, and may include **badSinceDate** to specify the time after which the certificate should not be trusted.

The **PKIProtection** field contains the requestor's signature, calculated on the DER encoded sequence of the header and body.

Revocation Response from CA to Requester

The CA will return a **RevRep** message to the requester. [8]

The PKIHeader includes the following information:

- **pvno** is zero;
- **messageType** is **RevRep**;
- **transactionID** is the same as the **transactionID** field in the **CertReq** message;
- **messageTime** is the current time with a granularity of minutes;
- **sender** is the distinguished name of the CA;
- **recipient** is the distinguished name of the ORA; and
- **protectionAlg** is the algorithm identifier for the signature algorithm used to protect the message.

The **PKIBody** is **RevRepContent**. If the CA revoked the certificate, the body will contain the following information:

- **status** will be **granted** or **grantedWithMods**; and
- **revDetails** will contain the **CertId** of the revoked certificate;

The **failinfo** field may not be present if **status** is **granted** or **grantedWithMods**.

If the CA rejected the request, the body shall include the following information:

- **status** will be **rejected**; and
- **failinfo** will contain the appropriate failure codes:
  - **badAlg** indicates that the CA cannot validate one of the signatures because the algorithm identifier is unrecognized or unsupported;
  - **badMessageCheck** indicates that the one of the signatures in the **PKIProtection** fields was checked but did not match;
  - **badRequest** indicates that the responder does not permit or support the transaction;
  - **badTime** indicates that the **messageTime** field in the message header was not sufficiently close to the responder's system time; or
  - **badCertId** indicates that the information in **latestCerts** did not identify an unexpired, unrevoked certificate.

If the certificate in question can be determined, **revDetails** will contain the **CertId** of the certificate whose revocation was rejected.

The **PKIProtection** field shall contain the CA's signature, calculated on the DER encoded sequence of the header and body.

6.5.4 Request Certificate from Directory

Entities may request certificates from the directory using LDAP and the matching rules certificate exact match and certificate match, as defined in [DAM].

[8] If the requester is an ORA, the CA may optionally send the **RevRep** message to the certificate holder as well.

6.5.5 Request Certificate Pair from Directory

Entities may request certificate pairs from the directory using LDAP and the certificate pair match rule, as defined in [DAM].

6.5.6 Request CRL from Directory

Entities may request CRLs from the directory using LDAP, the certificate list match rule, and the algorithm identifier match rule, as defined in [DAM].

6.5.7 Publish Certificate

CAs may publish a certificate they issued, or that was issued to them, in the directory.

For X.500 directory service, the CA will use DAP with strong authentication to bind to the directory.

6.5.8 Publish Cross Certificate

CAs may publish a cross certificates in the directory. Cross certificates may be added as attributes of the subject of the forward or reverse certificate.

For X.500 directory service, the CA will use DAP with strong authentication to bind to the directory.

6.5.9 Publish CRLs

CAs may publish CRLs in the directory.

For X.500 directory service, the CA will use DAP with strong authentication to bind to the directory.

7. References

[COR95] ISO/IEC JTC 1/SC 21/WG 4, *Technical Corrigendum 2 to ISO/IEC 9594-8 : 1990 & 1993 (1995:E)*. July 1995.

[DAM] ISO/IEC JTC 1/SC 21/WG 4, Draft Amendments DAM 4 to ISO/IEC 9594-2, DAM 2 to ISO/IEC 9594-6, DAM 1 to ISO/IEC 9594-7, and DAM 1 to ISO/IEC 9594-8 on Certificate Extensions, June 30, 1996.

[FIPS140] FIPS PUB 140-1, *Security Requirements for Cryptographic Modules*, NIST, January 1994.

[FIPS180] FIPS PUB 180-1, *Secure Hash Standard*, NIST, April 1995.

[FIPS186] FIPS PUB 186, *Digital Signature Standard*, NIST, May 1994.

[FIPS46] FIPS PUB 46-2, *Data Encryption Standard*, December 1993.

[FIPS81] FIPS PUB 81, *DES Modes of Operation*, NIST, December 1980.

[GCS] X/Open Preliminary Specification P442. June, 1996.

[ISO88] ISO/IEC 9594-8 (1988:E), *CCITT Information Technology - Open Systems Interconnection - The Directory: Authentication Framework.* Standard X.509, 1988.

[ISO94-6] ISO/IEC 9594-6 (1994), *Open Systems Interconnection - The Directory: Protocol Specifications*. 1994.

[ISO94-8] ISO/IEC 9594-8 (1994), *Open Systems Interconnection - The Directory: Authentication Framework*. 1994. The 1994 edition of this document has been amended by the Draft Amendments [DAM] and a *Technical Corrigendum* [COR95].

[PKIX1] Internet Draft, *Internet Public Key Infrastructure Part I: X.509 Certificate and CRL Profile*, R Housley, W. Ford and D. Solo, June 1996. working draft "in progress" available at: ftp://ds.internic.net/internet-drafts/draft-ietf-pkix-ipki-part1-02.txt

[PKIX3] Internet Draft, *Internet Public Key Infrastructure Part III: Certificate Management Protocols*, S. Farrell, C. Adams and W. Ford, working draft "in progress" available at: ftp://ds.internic.net/internet-drafts/draft-ietf-pkix-ipki3cmp-00.txt

[RFC822] RFC 822, *Standard for the Format of ARPA Internet Text Messages*, David H. Crocker, August 13, 1982

[RFC1777] RFC 1777, *Lightweight Directory Access Protocol*, Ed Yeoung, Howes, and Killie. March 1995.

[STAB95] OIW, *Stable Implementation Agreements for Open Systems Interconnection Protocols: Part 12 - OS Security*. June 1995.

[X9.55] Draft American National Standard X9.55-1995, *Public Key Cryptography for the Financial Services Industry: Extensions to Public Key Certificates and Certificate Revocation Lists*, Nov. 11, 1995

Appendix A - X.509 v3 Certificate ASN.1

```
AuthenticationFramework {joint-iso-ccitt ds(5) modules(1) authenticationFramework(7) 2}
DEFINITIONS ::=
BEGIN

-- EXPORTS All --
-- The types and values defined in this module are exported for use in the other ASN.1
--  modules contained within the Directory Specifications, and for the use of other applications
--  which will use them to access Directory services.  Other applications may use them for
 -- their own purposes, but this will not constrain extensions and modifications needed to
 -- maintain or improve the Directory service.

IMPORTS
        id-at, informationFramework, upperBounds selectedAttributeTypes,
                basicAccessControl FROM UsefulDefinitions {joint-iso-ccitt ds(5)
                modules(1) usefulDefinitions(0) 2}
        Name, ATTRIBUTE
                FROM InformationFramework informationFramework
        ub-user-password
                FROM UpperBounds upperBounds
        AuthenticationLevel
                FROM BasicAccessControl basicAccessControl
        UniqueIdentifier
                FROM SelectedAttributeTypes selectedAttributeTypes   ;

-- types --


Certificate             ::=     SIGNED {SEQUENCE{
        version                         [0]     Version DEFAULT v1,
        serialNumber                    CertificateSerialNumber,
        signature                       AlgorithmIdentifier,
        issuer                          Name,
        validity                        Validity,
        subject                         Name,
        subjectPublicKeyInfo                    SubjectPublicKeyInfo}
        issuerUniqueIdentifier          [1]     IMPLICIT UniqueIdentifier OPTIONAL,
                                                ---if present, version must be  v1 or v2--
        subjectUniqueIdentifier         [2]     IMPLICIT UniqueIdentifier OPTIONAL,
                                                ---if present, version must be v1 or v2--
        extensions                      [3]     Extensions Optional
                                                --if present, version must be v3--} }


 Version                        ::=     INTEGER {v1(0), v2(1), v3(2) }

CertificateSerialNumber         ::=     INTEGER
Algorithmidentifier             ::=     SEQUENCE{
        algorithm               ALGORITHM.&id({SupportedAlgorithms}),
        parameters              ALGORITHM.&Type  (SupportedAlgorithms}{ @algorithm}) OPTIONAL
        }
```

```
--      Definition of the following information object is deferred, perhaps to standardized
--      profiles of to protocol implementation conformance statements. This set is required to
--      specify a table constraint on the Parameters component of Algorithmidentifier.
--      SupportedAlgorithms ALGORITHM ::=       { ...|... }

Validity                        ::=     SEQUENCE{
        notBefore       UTCTime,
        notAfter        UTCTime}

SubjectPublicKeyInfo            ::=     SEQUENCE{
        algorithm               AlgorithmIdentifier,
        subjectPublicKey        BIT STRING}

Extensions              ::=     SEQUENCE OF Extension

Extension               ::=     SEQUENCE {
        extnId          EXTENSION.&id ({ExtensionSet}),
        critical        BOOLEAN DEFAULT FALSE,
        extnValue               OCTET STRING
                                -- contains a DER encoding of a value of type &ExtnType for the
                                -- extension object identified by extnId --

-- Definition of the following information object set is deferred, perhaps to
-- standardized profiles or to protocol implementation conformance statements.
-- The set is required to specify a table constraint on the critical component
-- of Extension.
--      ExtensionSet    EXTENSION       ::=     { ... | ... }

EXTENSION       ::=     CLASS
{
        &id             OBJECT IDENTIFIER UNIQUE,
        &ExtnType
}
WITH SYNTAX
{
        SYNTAX                  &ExtnType
        IDENTIFIED BY           &id
}

Certificates            ::=             SEQUENCE {
        certificate                     Certificate,
        certificationPath               ForwardCertificationPath OPTIONAL}

ForwardCertificationPath        ::=     SEQUENCE OF CrossCertificates

CertificationPath               ::=     SEQUENCE {
        userCertificate                 Certificate,
        theCACertificates               SEQUENCE OF CertificatePair OPTIONAL}

CrossCertificates               ::=     SET OF Certificate
```

```
CertificateList ::=                 SIGNED { SEQUENCE {
        signature                   AlgorithmIdentifier,
        issuer                      Name,
        thisUpdate                  UTCTime,
        nextUpdate                  UTCTime OPTIONAL,
        revokedCertificates         SEQUENCE OF SEQUENCE {
                userCertificate             CertificateSerialNumber,
                revocationDate              UTCTime,
                crlEntryExtensions          Extensions OPTIONAL } OPTIONAL,
        crlExtensions       [0]     Extensions OPTIONAL }}
CertificatePair     ::=     SEQUENCE {
        forward     [0]     Certificate OPTIONAL,
        reverse     [1]     Certificate OPTIONAL
                            -- at least one of the pair shall be present -- }

-- attribute types--

userPassword        ATTRIBUTE   ::=     {
        WITH SYNTAX             OCTET STRING (SIZE  (0..ub-user-password))
        EQUALITY MATCHING RULE  octetStringMatch
        ID                      id-at-userPassword }

userCertificate     ATTRIBUTE   ::=     {
        WITH SYNTAX             Certificate
        ID                      id-at-userCertificate }

cACertificate       ATTRIBUTE   ::=     {
        WITH SYNTAX             Certificate
        ID                      id-at-cACertificate }

authorityRevocationList     ATTRIBUTE   ::=     {
        WITH SYNTAX             CertificateList
        ID                      id-at-authorityRevocationList }

certificateRevocationList   ATTRIBUTE   ::=     {
        WITH SYNTAX             CertificateList
        ID                      id-at-certificateRevocationList }

crossCertificatePair    ATTRIBUTE   ::=     {
        WITH SYNTAX             CertificatePair
        ID                      id-at-crossCertificatePair }

-- information object classes --

ALGORITHM   ::=     TYPE-IDENTIFIER

-- Parameterized Types --
HASHED {ToBeHashed}         ::=     OCTET STRING ( CONSTRAINED-BY {
        --must be the result of applying a hashing procedure to the --
        --DER-encoded octets of a value of -- ToBeHashed })

ENCRYPTED { To\BeEnciphered}    :=      BIT STRING ( CONSTRAINED BY {
```

```
        --must be the result of applying an encipherment  procedure to the --
        --BER-encoded octets of a value of -- ToBeEnciphered })

SIGNED { ToBeSigned }       ::=     SEQUENCE{
        ToBeSigned,
        COMPONENTS OF SIGNATURE { ToBeSigned }),

SIGNATURE { OfSignature }   ::=     SEQUENCE {
        AlgorithmIdentifier,
        ENCRYPTED { HASHED { OfSignature }}}

-- object identifier assignments --

id-at-userPassword              OBJECT IDENTIFIER  ::=  {id-at 35}
id-at-userCertificate           OBJECT IDENTIFIER  ::=  {id-at 36}
id-at-cAcertificate             OBJECT IDENTIFIER  ::=  {id-at 37}
id-at-authorityRevocationList   OBJECT IDENTIFIER  ::=  {id-at 38}
id-at-certificateRevocationList OBJECT IDENTIFIER  ::=  {id-at 39}
id-at-crossCertificatePair      OBJECT IDENTIFIER  ::=  {id-at 40}
id-at-supportedAlgorithms       OBJECT IDENTIFIER  ::=  {id-at 52}
id-at-deltaRevocationList       OBJECT IDENTIFIER  ::=  {id-at 53}

END
```

Appendix B - Certificate and CRL Extensions ASN.1

```
CertificateExtensions {joint-iso-ccitt ds(5) module(1) certificateExtensions(26) 0}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
        id-at, id-ce, id-mr, informationFramework, authenticationFramework,
                selectedAttributeTypes, upperBounds
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1)
                usefulDefinitions(0) 2}
        Name, RelativeDistinguishedName, ATTRIBUTE, Attribute,
                MATCHING-RULE FROM InformationFramework informationFramework
        CertificateSerialNumber, CertificateList, AlgorithmIdentifier,
                EXTENSION
                FROM AuthenticationFramework authenticationFramework
        DirectoryString
                FROM SelectedAttributeTypes selectedAttributeTypes
        ub-name
                FROM UpperBounds upperBounds
        ORAddress
                FROM MTSAbstractService {joint-iso-ccitt mhs(6) mts(3)
                modules(0) mts-abstract-service(1) version-1994 (0) } ;

-- Unless explicitly noted otherwise, there is no significance to the ordering
-- of components of a SEQUENCE OF construct in this specification.

-- Key and policy information extensions --

authorityKeyIdentifier EXTENSION ::= {
        SYNTAX                  AuthorityKeyIdentifier
        IDENTIFIED BY           { id-ce 35 } }

AuthorityKeyIdentifier ::= SEQUENCE {
        keyIdentifier                   [0] KeyIdentifier               OPTIONAL,
        authorityCertIssuer             [1] GeneralNames                OPTIONAL,
        authorityCertSerialNumber       [2] CertificateSerialNumber     OPTIONAL }
        ( WITH COMPONENTS               {..., authorityCertIssuer PRESENT,
                                        authorityCertSerialNumber PRESENT} |
         WITH COMPONENTS {..., authorityCertIssuer ABSENT,
                                        authorityCertSerialNumber ABSENT} )

KeyIdentifier ::= OCTET STRING

subjectKeyIdentifier EXTENSION ::= {
        SYNTAX                  SubjectKeyIdentifier
        IDENTIFIED BY           { id-ce 14 } }

SubjectKeyIdentifier ::= KeyIdentifier

keyUsage EXTENSION ::= {
```

```
        SYNTAX          KeyUsage
        IDENTIFIED BY { id-ce 15 } }

KeyUsage ::= BIT STRING {
        digitalSignature                (0),
        nonRepudiation                  (1),
        keyEncipherment                 (2),
        dataEncipherment                (3),
        keyAgreement                    (4),
        keyCertSign                     (5),
        cRLSign                         (6) }

privateKeyUsagePeriod EXTENSION ::= {
        SYNTAX          PrivateKeyUsagePeriod
        IDENTIFIED BY { id-ce 16 } }

PrivateKeyUsagePeriod ::= SEQUENCE {
        notBefore       [0]     GeneralizedTime OPTIONAL,
        notAfter        [1]     GeneralizedTime OPTIONAL }
        ( WITH COMPONENTS           {..., notBefore PRESENT} |
        WITH COMPONENTS  {..., notAfter PRESENT} )

certificatePolicies EXTENSION ::= {
        SYNTAX          CertificatePoliciesSyntax
        IDENTIFIED BY { id-ce 32 } }

CertificatePoliciesSyntax ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation

PolicyInformation ::= SEQUENCE {
        policyIdentifier   CertPolicyId,
        policyQualifiers   SEQUENCE SIZE (1..MAX) OF
                PolicyQualifierInfo OPTIONAL }

CertPolicyId ::= OBJECT IDENTIFIER

PolicyQualifierInfo ::= SEQUENCE {
        policyQualifierId       CERT-POLICY-QUALIFIER.&id
                                        ({SupportedPolicyQualifiers}),
        qualifier               CERT-POLICY-QUALIFIER.&Qualifier
                                        ({SupportedPolicyQualifiers}{@policyQualifierId})
                                        OPTIONAL }

SupportedPolicyQualifiers CERT-POLICY-QUALIFIER ::= { ... }

CERT-POLICY-QUALIFIER ::= CLASS {
        &id             OBJECT IDENTIFIER UNIQUE,
        &Qualifier      OPTIONAL }
WITH SYNTAX {
        POLICY-QUALIFIER-ID         &id
        [QUALIFIER-TYPE     &Qualifier] }
```

```
policyMappings EXTENSION ::= {
        SYNTAX          PolicyMappingsSyntax
        IDENTIFIED BY { id-ce 33 } }

PolicyMappingsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        issuerDomainPolicy          CertPolicyId,
        subjectDomainPolicy  CertPolicyId }

supportedAlgorithms ATTRIBUTE ::= {
        WITH SYNTAX SupportedAlgorithm
        EQUALITY MATCHING RULE algorithmIdentifierMatch
        ID { id-at 52 } }

SupportedAlgorithm ::= SEQUENCE {
        algorithmIdentifier             AlgorithmIdentifier,
        intendedUsage                   [0]  KeyUsage OPTIONAL,
        intendedCertificatePolicies     [1]  CertificatePoliciesSyntax OPTIONAL }

-- Certificate subject and certificate issuer attributes extensions --

subjectAltName EXTENSION ::= {
        SYNTAX          GeneralNames
        IDENTIFIED BY { id-ce 17 } }

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
        otherName                       [0]     INSTANCE OF OTHER-NAME,
        rfc822Name                      [1]     IA5String,
        dNSName                         [2]     IA5String,
        x400Address                     [3]     ORAddress,
        directoryName                   [4]     Name,
        ediPartyName                    [5]     EDIPartyName,
        uniformResourceIdentifier       [6]     IA5String,
        iPAddress                       [7]     OCTET STRING,
        registeredID                    [8]     OBJECT IDENTIFIER }

OTHER-NAME ::= TYPE-IDENTIFIER

EDIPartyName ::= SEQUENCE {
        nameAssigner                    [0]     DirectoryString {ub-name} OPTIONAL,
        partyName                       [1]     DirectoryString {ub-name} }

issuerAltName EXTENSION ::= {
        SYNTAX          GeneralNames
        IDENTIFIED BY { id-ce 18 } }

subjectDirectoryAttributes EXTENSION ::= {
        SYNTAX          AttributesSyntax
        IDENTIFIED BY { id-ce 9 } }

AttributesSyntax ::= SEQUENCE SIZE (1..MAX) OF Attribute
```

```
-- Certification path constraints extensions --

basicConstraints EXTENSION ::= {
        SYNTAX          BasicConstraintsSyntax
        IDENTIFIED BY { id-ce 19 } }

BasicConstraintsSyntax ::= SEQUENCE {
        cA                      BOOLEAN DEFAULT FALSE,
        pathLenConstraint       INTEGER (0..MAX) OPTIONAL }

nameConstraints EXTENSION ::= {
        SYNTAX          NameConstraintsSyntax
        IDENTIFIED BY { id-ce 30 } }

NameConstraintsSyntax ::= SEQUENCE {
        permittedSubtrees       [0]     GeneralSubtrees OPTIONAL,
        excludedSubtrees        [1]     GeneralSubtrees OPTIONAL }

GeneralSubtrees ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree

GeneralSubtree ::= SEQUENCE {
        base                    GeneralName,
        minimum                 [0]     BaseDistance DEFAULT 0,
        maximum                 [1]     BaseDistance OPTIONAL }

BaseDistance ::= INTEGER (0..MAX)

policyConstraints EXTENSION ::= {
        SYNTAX          PolicyConstraintsSyntax
        IDENTIFIED BY { id-ce 34 } }

PolicyConstraintsSyntax ::= SEQUENCE SIZE (1..MAX) OF SEQUENCE {
        policySet               [0] CertPolicySet OPTIONAL,
        requireExplicitPolicy [1] SkipCerts OPTIONAL,
        inhibitPolicyMapping  [2] SkipCerts OPTIONAL }

SkipCerts ::= INTEGER (0..MAX)

CertPolicySet ::= SEQUENCE SIZE (1..MAX) OF CertPolicyId


-- Basic CRL extensions --

cRLNumber EXTENSION ::= {
        SYNTAX          CRLNumber
        IDENTIFIED BY { id-ce 20 } }

CRLNumber ::= INTEGER (0..MAX)

reasonCode EXTENSION ::= {
        SYNTAX          CRLReason
        IDENTIFIED BY { id-ce 21 } }
```

```
CRLReason ::= ENUMERATED {
        unspecified                 (0),
        keyCompromise               (1),
        cACompromise                (2),
        affiliationChanged          (3),
        superseded                  (4),
        cessationOfOperation        (5),
        certificateHold             (6),
        removeFromCRL               (8) }

instructionCode EXTENSION ::= {
        SYNTAX      HoldInstruction
        IDENTIFIED BY { id-ce 23 } }

HoldInstruction ::= OBJECT IDENTIFIER

invalidityDate EXTENSION ::= {
        SYNTAX      GeneralizedTime
        IDENTIFIED BY { id-ce 24 } }


-- CRL distribution points and delta-CRL extensions --

cRLDistributionPoints EXTENSION ::= {
        SYNTAX      CRLDistPointsSyntax
        IDENTIFIED BY       { id-ce 31 } }

CRLDistPointsSyntax ::= SEQUENCE SIZE (1..MAX) OF DistributionPoint

DistributionPoint ::= SEQUENCE {
        distributionPoint       [0]     DistributionPointName OPTIONAL,
        reasons                 [1]     ReasonFlags OPTIONAL,
        cRLIssuer               [2]     GeneralNames OPTIONAL }

DistributionPointName ::= CHOICE {
        fullName                [0]     GeneralNames,
        nameRelativeToCRLIssuer [1]     RelativeDistinguishedName }

ReasonFlags ::= BIT STRING {
        unused                      (0),
        keyCompromise               (1),
        caCompromise                (2),
        affiliationChanged          (3),
        superseded                  (4),
        cessationOfOperation        (5),
        certificateHold             (6) }

issuingDistributionPoint EXTENSION ::= {
        SYNTAX      IssuingDistPointSyntax
        IDENTIFIED BY       { id-ce 28 } }

IssuingDistPointSyntax ::= SEQUENCE {
        distributionPoint           [0] DistributionPointName OPTIONAL,
        onlyContainsUserCerts       [1] BOOLEAN DEFAULT FALSE,
```

```
        onlyContainsCACerts         [2] BOOLEAN DEFAULT FALSE,
        onlySomeReasons             [3] ReasonFlags OPTIONAL,
        indirectCRL                 [4] BOOLEAN DEFAULT FALSE }

certificateIssuer EXTENSION ::= {
        SYNTAX              GeneralNames
        IDENTIFIED BY       { id-ce 29 } }

deltaCRLIndicator EXTENSION ::= {
        SYNTAX              BaseCRLNumber
        IDENTIFIED BY       { id-ce 27 } }

BaseCRLNumber ::= CRLNumber

deltaRevocationList ATTRIBUTE ::= {
        WITH SYNTAX CertificateList
        EQUALITY MATCHING RULE certificateListExactMatch
        ID      {id-at 53 } }


-- Matching rules --

certificateExactMatch MATCHING-RULE ::= {
        SYNTAX              CertificateExactAssertion
        ID                  id-mr-certificateExactMatch }

CertificateExactAssertion ::= SEQUENCE {
        serialNumber        CertificateSerialNumber,
        issuer              Name }

certificateMatch MATCHING-RULE ::= {
        SYNTAX              CertificateAssertion
        ID                  id-mr-certificateMatch }

CertificateAssertion ::= SEQUENCE {
        serialNumber                [0] CertificateSerialNumber OPTIONAL,
        issuer                      [1] Name                    OPTIONAL,
        subjectKeyIdentifier        [2] SubjectKeyIdentifier    OPTIONAL,
        authorityKeyIdentifier      [3] AuthorityKeyIdentifier  OPTIONAL,
        certificateValid            [4] UTCTime                 OPTIONAL,
        privateKeyValid             [5] GeneralizedTime         OPTIONAL,
        subjectPublicKeyAlgID       [6] OBJECT IDENTIFIER       OPTIONAL,
        keyUsage                    [7] KeyUsage                OPTIONAL,
        subjectAltName              [8] AltNameType             OPTIONAL,
        policy                      [9] CertPolicySet           OPTIONAL,
        pathToName                  [10] Name                   OPTIONAL }

AltNameType ::= CHOICE {
        builtinNameForm     ENUMERATED {
                                rfc822Name          (1),
                                dNSName             (2),
                                x400Address         (3),
                                directoryName (4),
                                ediPartyName  (5),
```

```
                                uniformResourceIdentifier     (6),
                                iPAddress                     (7),
                                registeredId                  (8) },
        otherNameForm           OBJECT IDENTIFIER }

certificatePairExactMatch MATCHING-RULE  ::=      {
        SYNTAX                  CertificatePairExactAssertion
        ID                      id-mr-certificatePairExactMatch }

CertificatePairExactAssertion ::= SEQUENCE {
        forwardAssertion        [0] CertificateExactAssertion OPTIONAL,
        reverseAssertion        [1] CertificateExactAssertion OPTIONAL }
        ( WITH COMPONENTS               {..., forwardAssertion PRESENT} |
          WITH COMPONENTS               {..., reverseAssertion PRESENT} )

certificatePairMatch MATCHING-RULE ::=    {
        SYNTAX          CertificatePairAssertion
        ID              id-mr-certificatePairMatch }

CertificatePairAssertion ::= SEQUENCE {
        forwardAssertion        [0] CertificateAssertion OPTIONAL,
        reverseAssertion        [1] CertificateAssertion OPTIONAL }
        ( WITH COMPONENTS               {..., forwardAssertion PRESENT} |
          WITH COMPONENTS               {..., reverseAssertion PRESENT} )

certificateListExactMatch MATCHING-RULE  ::=      {
        SYNTAX                  CertificateListExactAssertion
        ID                      id-mr-certificateListExactMatch }

CertificateListExactAssertion ::= SEQUENCE {
        issuer                  Name,
        thisUpdate              UTCTime,
        distributionPoint       DistributionPointName OPTIONAL }

certificateListMatch MATCHING-RULE ::= {
        SYNTAX                  CertificateListAssertion
        ID                      id-mr-certificateListMatch }

CertificateListAssertion ::= SEQUENCE {
        issuer                  Name   OPTIONAL,
        minCRLNumber            [0]    CRLNumber    OPTIONAL,
        maxCRLNumber            [1]    CRLNumber    OPTIONAL,
        reasonFlags             ReasonFlags   OPTIONAL,
        dateAndTime             UTCTime       OPTIONAL,
        distributionPoint       [2]    DistributionPointName OPTIONAL }

algorithmIdentifierMatch MATCHING-RULE   ::=      {
        SYNTAX                  AlgorithmIdentifier
        ID                      id-mr-algorithmIdentifierMatch }


-- Object identifier assignments --

id-at-supportedAlgorithms               OBJECT IDENTIFIER  ::=      {id-at 52}
```

```
id-at-deltaRevocationList          OBJECT IDENTIFIER  ::=  {id-at 53}
id-ce-subjectDirectoryAttributes   OBJECT IDENTIFIER  ::=  {id-ce 9}
id-ce-subjectKeyIdentifier         OBJECT IDENTIFIER  ::=  {id-ce 14}
id-ce-keyUsage                     OBJECT IDENTIFIER  ::=  {id-ce 15}
id-ce-privateKeyUsagePeriod        OBJECT IDENTIFIER  ::=  {id-ce 16}
id-ce-subjectAltName               OBJECT IDENTIFIER  ::=  {id-ce 17}
id-ce-issuerAltName                OBJECT IDENTIFIER  ::=  {id-ce 18}
id-ce-basicConstraints             OBJECT IDENTIFIER  ::=  {id-ce 19}
id-ce-cRLNumber                    OBJECT IDENTIFIER  ::=  {id-ce 20}
id-ce-reasonCode                   OBJECT IDENTIFIER  ::=  {id-ce 21}
id-ce-instructionCode              OBJECT IDENTIFIER  ::=  {id-ce 23}
id-ce-invalidityDate               OBJECT IDENTIFIER  ::=  {id-ce 24}
id-ce-deltaCRLIndicator            OBJECT IDENTIFIER  ::=  {id-ce 27}
id-ce-issuingDistributionPoint     OBJECT IDENTIFIER  ::=  {id-ce 28}
id-ce-certificateIssuer            OBJECT IDENTIFIER  ::=  {id-ce 29}
id-ce-nameConstraints              OBJECT IDENTIFIER  ::=  {id-ce 30}
id-ce-cRLDistributionPoints        OBJECT IDENTIFIER  ::=  {id-ce 31}
id-ce-certificatePolicies          OBJECT IDENTIFIER  ::=  {id-ce 32}
id-ce-policyMappings               OBJECT IDENTIFIER  ::=  {id-ce 33}
id-ce-policyConstraints            OBJECT IDENTIFIER  ::=  {id-ce 34}
id-ce-authorityKeyIdentifier       OBJECT IDENTIFIER  ::=  {id-ce 35}
id-mr-certificateExactMatch        OBJECT IDENTIFIER  ::=  {id-mr 34}
id-mr-certificateMatch             OBJECT IDENTIFIER  ::=  {id-mr 35}
id-mr-certificatePairExactMatch    OBJECT IDENTIFIER  ::=  {id-mr 36}
id-mr-certificatePairMatch         OBJECT IDENTIFIER  ::=  {id-mr 37}
id-mr-certificateListExactMatch    OBJECT IDENTIFIER  ::=  {id-mr 38}
id-mr-certificateListMatch         OBJECT IDENTIFIER  ::=  {id-mr 39}
id-mr-algorithmIdentifierMatch     OBJECT IDENTIFIER  ::=  {id-mr 40}

-- The following OBJECT IDENTIFIERS are not used by this specification:
-- {id-ce 2}, {id-ce 3}, {id-ce 4}, {id-ce 5}, {id-ce 6}, {id-ce 7},
-- {id-ce 8}, {id-ce 10}, {id-ce 11}, {id-ce 12}, {id-ce 13},
-- {id-ce 22}, {id-ce 25}, {id-ce 26}

END
```

Appendix C. API for Certificate Processor

This section defines a high level API for DoE Travel and other applications which are not cryptographically aware.

C.1 Common Data Structures

The certificate processing module shall support an ASN.1 encoded signed message format and status codes as common parameters.

```
SignedMessage ::=  SEQUENCE {
        message                 CompoundMessage,      -- time/dated message with signer's
                                                      -- name and certificate number
        signatureValue  OCTET STRING }        -- signature calculated on message

CompoundMessage ::= SEQUENCE {
        body                    OCTET STRING,         -- message to be signed
        timestamp               UTCTtime,             -- time object was signed
        signers_name            Name,                 -- distinguished name of signer
        cert_issuer             Name,                 -- distinguished name of certificate
                                                      -- issuer
        cert_num                INTEGER }             -- signer's certificate serial number
```

The certificate processing module shall support the following type definitions:

```
#typedef        DN      *char;          /* DN is an X.500 distinguished name.  It is represented
                                                as an alphanumeric string
        */
#typedef        time    *char;          /* string of numeric characters representing time and
                                          date. The string is interpreted as ddmmyyyyhhxx
                                          where
                                                dd is the day of the month
                                                mm is the month
                                                yyyy is the year
                                                hh is the hour ("military time", e.g., at 1:15 PM
                                                        hh would be  "13")
                                                xx is th e number of minutes past the hour (e.g.,
                                                        for 1:15 PM, xx would be "15")
                                          Note that seconds are ommitted.
        */
#typedef        SN      *char;          /* SN is a serial number associated with a certificate.
                                                It is represented by a numeric string.
        */
#typedef        CRL     *char;          /* CRL is a pointer to an array of characters.  The
                                          array contains an ASN.1 encoded X.509 certificate
                                          revocation list                                 */
#typedef        CERT    *char;          /* CERT is a pointer to an array of characters.  The
                                          array contains an ASN.1 encoded X.509 certificate. */
```

The certificate processing module shall support the following constants:

```
GENERAL_FAILURE     10h     call failed; reasons unknown or unspecified
GENERAL_ALLOC       11h     memory allocation failed, subroutine terminated.
GENERAL_PARAMS      12h     parameters missing or ill-formed
```

Initialization Status Codes		
CM_INIT	00h	initialization completed
CM_INIT_FAILED	01h	initialization failed
CM_INIT_NOCACHE	02h	initialization completed; no certificate or CRL cache
CM_INIT_NOINFO	03h	initialization completed; no local information
CM_INIT_NEITHER	04h	initialization completed; no cache or local information

Login Status Codes		
CM_LOGIN	00h	login completed
CM_LOGIN_BADPASS	01h	login failed; bad password

Logout Status Codes		
CM_LOGOUT	00h	signed local information and saved cache
CM_LOGOUT_NOSIG	01h	could not sign local information
CM_LOGOUT_FAILED	02h	logout failure

Compute Signature Status Codes		
CM_CS	00h	signed message
CM_CS_NOMSG	01h	no message to sign
CM_CS_BADTIME	02h	bad timestamp
CM_CS_NONAME	03h	user name unavailable
CM_CS_NOCERT	04h	user certificate number unavailable
CM_CS_ALLOC	05h	could not allocate required memory

Verify Signature Status Codes		
CM_VS	00h	verified message
CM_VS_BADCON	01h	ill-formed constraints
CM_VS_BADTIME	02h	bad timestamp
CM_VS_CONVIOL	03h	path(s) violate constraints
CM_VS_NOCERT	04h	user certificate unavailable
CM_VS_CERTEXP	05h	user certificate expired before signature was generated (according to the timestamp in the message)
CM_VS_NOPATH	06h	could not find path to Travel System CA

CA.2 Subroutine Calls

C.2.1 General Application Subroutine Calls

cm_initialize(Status:int)

Parameters:

Status output {CM_INIT, CM_INIT_*, GENERAL_*} CM_INIT indicates successful completion; CM_INIT_FAILED indicates the module failed to initialize; other CM_INIT_* codes indicate that the module does not have complete information, but is available for selected operations.

Functionality:

The cm_initialize function prepares the certificate processing module to perform basic operations, but does not decrypt a private key. The initialization function will load the certificate and CRL cache, as well as the local information such addresses of PKI

components (e.g., the CA and DS). If the cache or local information is not present, the module will function, but will return appropriate status codes. This permits the user to generate their keys and certificate request on the module.

After the intialization function is called, the cryptographic module shall accept calls to verify signatures, generate hashes, login the user, and get certificates or CRLs. Any other commands require a private DSA key, and will require require a cm_login command.

cm_login(Password:*char,Status:int)

Parameters:

Password: input pointer to a string containing an alphanumeric/special character text string of unlimited length representing the user's password.

Status: output status/error code {CM_LOGIN, CM_LOGIN_BADPASS, GENERAL_*} Only CM_LOGIN indicates successful completion.

Functionality:

A cm_login command will result in retrieval/loading and decryption of the private DSA key, if a key has been previously generated. The module will verify that the private key has been recovered by generating and comparing signatures on locally stored information.

cm_logout(Status:int)

Parameters:

Status: output status/error codes. Legal values are {CM_LOGOUT, CM_LOGOUT_*, GENERAL_*} Only CM_LOGOUT indicates successful completion.

Functionality:

The cm_logout command will return the associated cryptographic module to its unitialized state. Software cryptographic modules will additionally overwrite the memory used by the crypto module, to prevent disclosure of the user's private key through re-allocation of memory. The erasure will comply with ANSI X.9-17-1985 requirements for zeroizing Random Access Memory.

cm_compute_sig(Dlen:int, Data:*char, Current_time:TIME, Signedmsg:SMSG, Status:int)

Parameters:

Dlen	input	the length, in bytes, of the input data.
Data	input	data to be timestamped and signed.
Current_time	input	current time; will be inserted into message before signing.

Signedmsg output ASN.1 encoded, signed message; will incude user name, certifcate serial number, and time signature was applied.

Status output status/error codes. Legal values are {CM_CS, CM_CS_*, and GENERAL_*}. Only CM_CS indicates successful completion.

Functionality:

The compute signature command recieves a pointer to a variable length block of data, the data length and the time. The module will use the locally stored user Name and public key to generate a signed message conforming to the format given above.

cm_verify_sig(Signed_MSG:SMSG, Constraints:&char[], Current_time:TIME, Status:int)

Parameters:

Signed_Msg input signed message that requires verification

Constraints input pointer to an ASN.1 encoded sequence of certification path constraints; legal values are `basicConstraints` and `policyConstraints`.

Current_time input current date and time; required to determine if locally cached CRLs are still current

Status output error code/output {CM_VS, CM_VS_*, GENERAL_*}. Only CM_VS indicates successful verification of the signature and path.

Functionality:

cm_verify_sig receives three parameters: a pointer to a SignedMessage as defined in A.1, constraints upon the verification path, and a pointer to a status field. Verify_Signature will use the information in the SignedMessage to find the certificates and CRLs required to accept or reject the signature. cm_verify_sig will retrieve certificates and CRLs from the local cache or DS as required. Once a valid path to the Travel System CA has been determined, the DSS parameters can be determined for each set of signatures. Validity of a path shall be evaluated according to the guidelines presented in Section 6.3. Signatures are then calculated for each certificate and CRL as well as the message itself.

Cm_verify_sig returns a status code with the result. The status code CM_VS indicates that the signature and certificate path were both verified. CM_BADTIME indicates that the certificate or path were not valid at the time the signature was applied to the SignedMessage. Other error messages indicate that the user certificate could not be found, had expired, or a valid path to the Travel System CA could not be constructed.

C.2.2 Infrastructure-Related Operations

Revoke_Cert(Reason:int, Time:TIME, Status:int)

Parameters:

Reasons: input. Legal values are {keyCompromise, affiliationChanged, supersed, cessationOfOperation, unspecified}

Time: input. This parameter signifies the last time signatures associated with this certificate should be trusted.

Status: output. This parameter provides status/error codes. Legal values are {CM_REQ, CM_REQ_*, and GENERAL_*}. Only CM_REQ indicates successful completion.

Functionality:

Revoke_Cert will revoke the current user's certificate, for the reason supplied as parameter Reason. The module will generate a Cert_Revoke_Req message, sign it with the corresponding private key, and send it to the CA. The module will wait for a response for 300 seconds.

Gen_Cert_Req(Current_time:TIME,Status:int)

Parameters:

Current_time input current date and time

Status output status/error codes. Legal values are {CM_REQ, CM_REQ_*, and GENERAL_*}. Only CM_REQ indicates successful completion.

Functionality:

Generate certificate request - This command will generate a new public-private key pair, create a certificate request message, sign it with the private key and will write it to a diskette for delivery to an ORA.

Renew_Cert(Current_time:TIME,Status:int)

Parameters:

Current_time input current date and time

Status output status/error codes. Legal values are {CM_REN, CM_REN_*, and GENERAL_*}. Only CM_REN indicates successful completion.

Functionality:

Renew certificate will generate a new certificate request and sign it with the new private key and the old private key. The certificate request will be a `KeyUpdReq` message. It will be doubly signed, as defined in Section 6.4.1. The inner message will be signed with

the private key associated with the new public key. The outer message will be signed with the private key associated with the user's current certificate.

The KeyUpdReq message will contain the information noted in Section 6.4.3.

The doubly wrapped message will be sent to the CA as an electronic mail message.

Get_Cert_by_Name(User_name:DN, Number:int, Certificates:&Cert[], Status:int)

Parameters:

User_name: input	distinguished name of the user whose certificates are needed
Number: output	number of certificates retrieved
Certificates: output	array of *Number* certificates for User_name
Status: output	status codes (CM_CNAME, CM_CNAME_*, or GENERAL_*)

Functionality:

Get_Cert_by_Name will check the local cache for certificates associated with a particular user name. If no matching certificates are found in the cache, the X.500 DUA will request all certificates for the specified distinguished name from the directory service.

When matching certificates are found, space is allocated for the array *Certificates*, and the matching certificates are returned to the user. The number of certificates in the array is passed in the parameter *Number*. Get_Cert_by_Name may also store the retrieved certificate(s) in the local cache.

If the certificate is not found, Get_Cert_by_Name will return the status code CM_CNAME_NOT. If the directory service is unavailable, the status code CM_CNAME_NODS will be returned.

Get_Cert_by_Number(Cert_serial_number:SN, Issuer_name:DN, Certificate:CERT,Status:int)

Parameters:

Cert_serial_number: input	serial number of requested certificate
Issuer_name: input	issuer's distinguished name
Certificate: output	certificate with serial number equal to Cert_serial_number issued by Issuer_name
Status: output	status/error code: legal values are {CM_CNUM, CM_CNUM_*, an GENERAL_*} Only CM_CNUM indicates normal completion.

Functionality:

Get_Cert_by_Number will check the local cache for a certificate withe the specified serial number issued by the specified CA. If the certificate is not found in the cache, the X.500 DUA will request the certificate for the specified serial number and issuer name from the directory service. Get_Cert_by_Number may also store the retrieved certificate in the local cache.

When the certificate is found, space is allocated for the parameter *Certificate*, and the certificate is returned to the user.

If the certificate is not found, Get_Cert_by_Number will return the status code CM_CNUM_NOT. If the directory service is unavailable, the status code CM_CNUM_NODS will be returned.

get_CRL(CRL_name:DN, Rev_list:CRL, Status:int)

Parameters:

CRL_name: input Distinguished name of CRL

Rev_list: output CRL

Status: output status/error codes: Legal values are {CM_CRL, CM_CRL_*, and GENERAL_*}; only CM_CRL indicates successful completion.

Functionality:

Get_CRL will request the current certificate revocation list for a particular certification authority from the Directory Service. It may also store the retrieved CRL in the local cache.

Remove_Private_Key(Status:int)

Parameters:

Status: output This parameter conveys status/error codes. Legal values are {CM_RKEY, CM_RKEY_*, GENERAL_*}; only CM_RKEY indicates successful completion.

Functionality:

This command will delete the private key from the module, including its storage areas. All erasures will comply with military standards for secure erasure.

Appendix D. Cryptographic Application Programming Interface

The programming interface for the cryptographic module shall be a subset of the Generic Cryptographic Services API (GCS-API) from X/Open [GCS]. The programming interface shall conform to the X/Open *Preliminary Specification P442*, June 1996.[9]

The cryptographic module shall support cryptographic contexts implementing the algorithms and modes listed in Table B-1.

Algorithm	Mode	Key Size
DSS with SHA-1	sign, verify	1024
DES	ECB	56

The cryptographic module shall support the GCS-API Cryptographic Service Facility model. The cryptographic module shall meet all functional requirements identified in Section 5; the precise set of functions required shall be selected by the vendor from [GCS].

[9] This specification is available from X/Open. The document reference is X/Open Preliminary Specification P442 ISBN 1-85912-195-0 (6/96).

Appendix E. Sample Performance Specifications

This section provides sample performance specifications for PKI components. The actual performance requirements for DoE Travel should be determined according to the application and the computing platforms for each component. The performance requirements provided in Section F.4 are designed for a software cryptographic module on a 33 Mhz 486 PC.

E.1 CA Performance Specifications

Table E-1 provides sample performance specifications for CA functions. The response times are for local processing and exclude communications delays and human review or intervention, and are measured from the time the last byte of a request transaction is received at the CA to the time the first byte of response leaves the CA.

Table E-1 Performance Specifications for CA Functions

Function	Inputs	Outputs	Response Time
generate own public private key	size of modulus p	public and private keys	20 seconds
process certificate request	subject distinguished name, p, q, g, y	certificate to DS and requesting ORA	30 seconds
process request to revoke a certificate	certificate number or other unique identifier	CRL to DS; return signed acknowledgement	20 seconds

E.2 ORA Performance Specifications

Table E-2 provides sample performance specifications for ORA functions. These specifications include maximum response time and availability requirements. The response times are maximum response times under the estimated workload, excluding communications delays. They are measured from the time the last byte of a request transaction is received at the ORA to the time the first byte of response leaves the ORA. Reliability requirements (e.g., mean time between failures (MTBF), mean time to recover (MTTR), etc.) are equivalent to those for commercial computer systems operating in an office environment.

Table E-2 Performance Specifications for ORA Functions and Transactions

Function/ Transaction	Inputs	Outputs	Response Time
generate a key pair	size of p	p, q, g, x, y.	30 seconds
request certificate	p, q, g, and y	certificate request to CA	30 seconds
request certificate revocation	certificate number	revocation request to CA	30 seconds

E.3 Certificate Processing Performance Specifications

Table E-3 provides sample performance specifications for certificate processing functions. These specifications include maximum response time for generation of transactions. These times are measured from the time the certificate processing module is called to the time the transaction is generated and the communications module is called.

Table E-3 Performance Specifications for a Certificate Processing Module (All Functions and Transactions are Local)

Functions/ Transaction	Inputs	Outputs	Response Time
request certificate registration	p, q, g, and y	certificate request to ORA	45 seconds
electronic certificate request	p, q, g, and y	certificate request to ORA/CA	30 seconds
request certificate revocation	certificate number	revocation request to CA	30 seconds
request a certificate from DS	subject DN	request to DS	10 seconds
request a CRL from DS	CA DN	request to DS	10 seconds
path verification (length 2)	sender DN	sender public key	45 seconds

E.4. Software Cryptographic Module Performance Specifications

Table E-4 provides sample performance specifications for a software cryptographic module. These specifications assume a 486 33Mhz PC platform and do not use performance acceleration techniques such as pre-computation for DSS digital signatures. A system using pre-computation might exhibit a longer initialization times, but enhanced signature generation performance.

Table E-4 Performance Specifications for a Software Cryptographic Module (All Functions and Transactions are Local)

Functions/ Transactions	Inputs	Outputs	Response Time
generate DSA key pair	p, q, g	y (x held internally)	45 seconds
generate SHA-1 hash	message	message digest	5 seconds (5K byte message)
signature generation	message digest and (p, q, g, y)	DSS signature	45 seconds
signature verification	message, signer's public key and (p,q,g)	binary	1 minute (5K byte message)
load private key	password	status	20 seconds
export public key	password	encrypted private key	20 seconds
initialize module	NA	status	15 seconds

E.5 DS Performance Specifications

Table E-5 provides the performance specifications for DS functions. These specifications include response time and availability requirements. The response times are maximum response times under the specified workload. The response times exclude communications delays. They are measured from the time the last byte of request transaction is received at the DS to the time the first byte of response leaves the DS or the transaction processing is completed.

Table E-5 Performance Specifications for Directory Service Transactions

(All transactions are electronic)

Transaction	**Inputs**	**Outputs**	**Response Time**
add a certificate	certificate	none	10 seconds
post new CRL	CRL	none	10 seconds
certificate request	certificate number and issuer DN, or owner DN	certificate or error message	10 seconds
CRL request	issuer DN	CRL	10 seconds

Note: The issuer distinguished name (DN) is redundant because there is only one CA for the Travel System, but it is necessary to allow for future enhancements.

Appendix F. CA Key Pair Update

Like any PKI entity, the Travel CA's certificate will need to generate a new key pair periodically. The text here provides a mechanism where key pair update can be achieved without requiring all users to immediately visit an ORA. This procedure uses the current key pair to protect the new key pair, so users who trust the old key pair can trust the new certificate.

This text is drawn from the current draft of the PKI-X specifications.

F.1 Overview and Rationale

The basis of the procedure described here is that the CA protects its new public key using its previous private key and vice-versa. Thus when a CA updates its key pair it must generate two new cACertificate attribute values if certificates are made available using an X.500 directory.

When the CA changes its key pair those entities who have acquired the old CA public key via "out-of-band" means are most affected. These end entities begin all certification paths with the old CA key pair. It is these end entities who will need access to the new CA public key. By protecting it with the old CA private key, these end entities can construct a certification chain for certificates issued under the new key pair.

When the user returns to an ORA for a new certificate, they will acquire the new certificate through out-of-band means, and will be able to trust that key pair directly.

The data structure used to protect the new and old CA public keys is a standard certificate (which may also contain extensions). There are no new data structures required.

F.2 CA Operator actions

To change the key of the CA, the CA operator does the following:

1. Generate a new key pair.
2. Create a certificate containing the old CA public key signed with the new private key (the "old with new" certificate).
3. Create a certificate containing the new CA public key signed with the old private key (the "new with old" certificate).
4. Create a certificate containing the new CA public key signed with the new private key (the "new with new" certificate).
5. Publish these new certificates via the directory and/or other means. (A CAKeyUpdAnn message.)
6. Export the new CA public key so that end entities may acquire it using the "out-of-band" mechanism.

The old CA private key is then no longer required. The old CA public key will however remain in use for some time. The time when the old CA public key is no longer required (other than for non-repudiation) will be when all end entities of this CA have acquired the new CA public key via "out-of-band" means.

The "old with new" certificate should have a validity period starting at the generation time of the old key pair and ending at the time at which the CA will next update its key pair.

The "new with old" certificate should have a validity period starting at the generation time of the new key pair and ending at the time by which all end entities of this CA will securely possess the new CA public key.

The "new with new" certificate should have a validity period starting at the generation time of the new key pair and ending at the time at which the CA will next update its key pair.